

前言

在过去几年中, JavaScript 发生了翻天覆地的变化。它曾经是一种处于次要地位的“玩具”语言, 但是现在 JavaScript 已经成为世界上最重要的程序设计语言之一。随着基于 Ajax 的开发的的重要性不断提升, 以及各种功能完备的 JavaScript 库的出现, 围绕在 JavaScript 周围的魔咒已经一扫而空。jQuery 无疑是最流行的也是对初学者最友好的 JavaScript 库, 它是 JavaScript 库中的佼佼者。

jQuery 不仅仅是初学者的最佳选择, 它已经在世界上最大的组织机构中得以应用, 每个月都为数以亿计的页面访问增强了交互性。Amazon、IBM、Twitter、NBC、Best Buy 和 Dell 等大量公司都在其产品中使用了 jQuery。

根据 Web 的发展规律, 毫无疑问 jQuery 将以 Web 发展的速度迅速演化。2011 年至少发布了 3 个主要的 jQuery 版本, 围绕 jQuery 的开发社区不断发展壮大, 全世界的开发人员都为 jQuery 的 bug 修正、插件开发和与之相关 jQuery UI 和 QUnit 等项目作出了重要的贡献。对于任何想从事世界级 JavaScript 开发的程序人员而言, 活跃的 jQuery 技术社区确保 jQuery 成为一个功能完备的 JavaScript 库。

无论采用哪一种程序设计方法学或编程技术, 对于 Java/Spring、PHP、.NET、Ruby on Rails 和 Python/Django 等各种 Web 技术, jQuery 在前端开发领域都具有突出的特色。

如果读者具有 HTML、CSS 和 JavaScript 的开发经验, 那么本书非常适合你阅读。本书将着重介绍 jQuery 核心库, 扩展你对 jQuery 的认识, 此外随之介绍的强大的核心 JavaScript 专业知识, 也为本书增色不少。本书的前面几章将帮助读者建立 jQuery 的开发环境、回顾重要的 JavaScript 概念。从第 3 章~第 7 章将着重介绍 jQuery 的核心概念。本书第 II 部分将着重介绍在实际开发中 jQuery 的应用, 详细介绍 jQuery UI、插件开发、jQuery 模板、单元测试、最佳实践以及在 jQuery 中应用的设计模式。

希望本书能帮助读者学会 jQuery 的硬功夫, 以便解决 Web 开发中所面临的各种问题。

读者对象

本书适合以下三种读者阅读:

- 富有经验的服务器端 Web 应用程序开发人员, 正在努力地迁移到客户端编程, 并想使用世界上最流行的前端库。
- 富有 JavaScript 程序设计经验的开发人员, 正想快速地掌握 jQuery。

- 普通 jQuery 使用者或中级 jQuery 开发人员,正希望扩展 jQuery 的知识、掌握 jQuery 的高级内容。

本书并不是为新手编写的。对于刚开始学习 HTML、CSS 和 JavaScript/jQuery 开发的新手来说,参阅 Richard York 编写的 *Beginning JavaScript and CSS Development with jQuery* 一书是一种更合适的选择。

内容提要

本书从开发人员的层次对 jQuery 提供了一个全面的介绍。另外还深入介绍了 jQuery 的很多高级特性。

在本书第 I 部分中,深入介绍了 jQuery 的基础知识、介绍了如何选择或操作 DOM 元素,以及如何绑定并处理浏览器的事件。

在掌握了坚实的基础知识之后,本书将继续介绍 jQuery 的一些高级主题,包括使用 JavaScript 进行插件开发、单元测试和 jQuery 库的其他一些高级特性。

本书着重介绍 jQuery 1.7.1 的可用特性,但在相关的地方都要注意在低版本 jQuery 库中的支持能力。

内容结构

本书分为两个部分,第 I 部分是 jQuery 基础,第 II 部分是 jQuery 应用。在 jQuery 基础部分中介绍了 jQuery 的核心概念,在 jQuery 应用部分中则着重介绍了一些高级主题。

第 I 部分——jQuery 基础,这一部分包含了下列章节:

第 1 章“jQuery 入门”——这一章介绍了如何建立开发和调试 jQuery 和 JavaScript 代码所需的环境,定义了在本书中使用的编码规范和标准。还介绍了将 JavaScript 代码打包作为产品并创建代码的方法。

第 2 章“JavaScript 基础”——这一章介绍了 JavaScript 程序设计语言的基础知识,为本书后面的内容打好一个基础。毕竟, jQuery 是一个 JavaScript 库,它的很多优秀特性来源于对 JavaScript 核心技术的巧妙应用。

第 3 章“jQuery 核心技术”——这一章介绍了 jQuery 库的基本功能。它描述了核心 jQuery 函数的使用方法,介绍了很多有用的工具函数,使用这些工具函数可以执行各种各样的编程任务。

第 4 章“选择和操作 DOM 元素”——这一章深入介绍了 jQuery 的核心特性,即选择或操作 HTML 元素的功能。

第 5 章“事件处理”——这一章介绍了 jQuery 的另外一个关键特性,即跨浏览器的事件绑定和事件管理功能。

第 6 章“HTML 表单、数据和 Ajax”——这一章介绍了过去 10 年以来 Web 开发领域

最大的技术变革——Ajax。

第7章“动画和特效”——这一章介绍了jQuery为页面组件创建动画效果的一些快捷方法，比如移动(move)、淡化(fade)、切换(toggle)和缩放(resize)。

第II部分——jQuery应用，包含了下面几个主题：

第8章“jQuery UI 第I部分：更轻松地创建Web界面”——这一章介绍了jQuery UI。jQuery UI是一个jQuery的用户界面库，它包含了一些部件(widget)、特效、动画和交互功能。

第9章“jQuery UI 第II部分：鼠标交互”——这一章介绍了jQuery UI鼠标操作的特性，包括移动、排序、缩放和使用鼠标选取元素。

第10章“编写高效的jQuery代码”——这一章介绍了大量jQuery优化技术、最佳实践和设计模式。可以立即将这些技术应用于你的代码中，使代码更清晰、更高效，具有更好的可维护性。

第11章“jQuery模板”——这一章专门介绍了jQuery Template插件。jQuery模板是一种将数据与标记代码融合的标准方法。

第12章“编写jQuery插件”——这一章重点介绍了如何开发jQuery插件。开发人员可以创建自定义方法以扩展jQuery功能，对于顶尖的jQuery开发人员来说这是一种基本能力。

第13章“使用jQuery Deferred对象进行高级异步编程”——这一章介绍了jQuery的Deferred对象，即\$.Deferred对象。它是在jQuery 1.5版本中引入的一个可链式调用的工具对象，它为回调函数的处理方式提供了精细的控制能力。

第14章“使用QUnit进行单元测试”——这一章介绍了单元测试的常见概念，并详细介绍了jQuery项目自身创建并使用的单元测试框架QUnit。

学习本书的前提条件

jQuery支持下列Web浏览器。读者只需要具有其中一种浏览器就可以运行本书中的例子。

- Firefox 3.6，或最新版本的Firefox浏览器
- Internet Explorer 6+
- Safari 5.0.x
- Opera 最新版本
- Chrome 最新版本

本书约定

为了更清楚地向读者说明文本的含义和文本格式的用途，本书使用了以下一些规范。



警告：带有类似警告图标的文本框表示与周围内容直接有关的、重要的、应该记住的信息。



提示：铅笔图标表示注意、提示、技巧和对当前所讨论内容的旁注。

本书中文本样式的规范：

- 在介绍时高亮显示新的术语和重要的词汇
- 采用类似的方式表示键盘组合键：Ctrl+A
- 在文本内以这种方式显示文件名、URL 和代码：persistence.properties
- 本书采用以下两种方式表示代码：

We use a monofont type with no highlighting for most code examples.

We use bold to emphasize code that is particularly important in the present context or to show changes from a previous code snippet.

源代码

在学习本书的过程中，读者既可以手工输入所有的代码，也可以使用本书配套的源代码文件。本书使用的所有源代码文件都可以从 www.wrox.com 网站下载。在访问该网站时，只需要简单地查找本书的书名标题(使用搜索框或书籍列表)，在本书的详细信息页面上点击 Download Code 链接，就可以下载到本书的全部源代码。可以下载的源代码在本书中用下面的图标突出显示：



在代码清单的标题中包含了源代码的文件名，如果仅仅是一个代码片段，则可以从代码的旁注中找到文件名，比如：

代码片段 filename



提示：由于很多图书的名称都很类似，使用 ISBN 更容易找到本书。本书的 ISBN 为：978-1-118-02668-7。

在下载了源代码之后,只需要使用你最常用的压缩软件对其解压即可。另外,也可以访问 Wrox 的代码下载页面 www.wrox.com/dynamic/books/download.aspx,查找本书或所有 Wrox 书籍可供下载的源代码。

勘误表

尽管我们竭尽所能来确保在正文和代码中没有错误,但人无完人,错误在所难免。如果你在 Wrox 出版的书中发现了错误(例如拼写错误或代码错误),我们将非常感谢你的反馈信息。发送勘误表将节省其他读者的时间,同时也会帮助我们提供更高质量的信息。

要找到本书的勘误页面,可以进入 <http://www.wrox.com>,使用 Search 搜索框或书名列表定位本书,然后在 Book Search Results 页面上单击 Errata 链接。在这个页面上可以查看为本书提交的、Wrox 编辑粘贴上去的所有勘误信息。完整的书名列表(包括每本书的勘误表)也可以从 www.wrox.com/misc-pages/booklist.shtml 上获得。

如果你在本书的勘误页面上没有看到你发现的错误,那么请单击 Errata Form 链接填写表单,把你发现的错误发给我们。我们会检查这些信息,如果属实,就把它添加到本书的勘误页面上,并在本书随后的版本中更正错误。

另外,在阅读本书的过程中,如果您有任何意见,或者想要发表自己的看法,请通过邮箱 wkservice@vip.163.com 与我们联系。

p2p.wrox.com

如果想和作者或同行进行讨论,请加入 p2p.wrox.com 上的 P2P 论坛。该论坛是一个基于 Web 的系统,你可以发布有关 Wrox 图书及相关技术的消息,与其他读者或技术人员交流。该论坛提供了订阅功能,当你感兴趣的主题有新帖子发布时,系统会向你发送邮件。Wrox 的作者、编辑、其他业界专家和像你一样的读者都会出现在这些论坛中。

在 <http://p2p.wrox.com> 网站上,你会找到很多不同的论坛,它们不但有助于你阅读本书,还有助于你开发自己的应用程序。加入论坛的步骤如下:

- (1) 进入 p2p.wrox.com,单击 Register 链接。
- (2) 阅读使用条款,然后单击 Agree 按钮。
- (3) 填写加入该论坛必需的信息和其他你愿意提供的信息,单击 Submit 按钮。
- (4) 你将收到一封电子邮件,描述如何验证你的账户和完成加入过程。



注意: 不加入 P2P 也可以阅读论坛里的消息。但是如果要发布自己的消息,就必须加入论坛。

加入论坛后,就可以发布新的消息和回复其他用户发布的消息。可以随时在 Web 上阅读论坛里的消息。如果想让某个论坛的新消息以电子邮件的方式发给你,可以单击论坛列

表中论坛名称旁边的 **Subscribe to this Forum** 图标。

要了解如何使用 Wrox P2P 的更多信息，请阅读 **P2P FAQ**，其中回答了论坛软件如何使用的问题，以及许多与 P2P 和 Wrox 图书相关的问题。要阅读 FAQ，单击任何 P2P 页面上的 FAQ 链接即可。



目 录

第 I 部分 jQuery 基础

第 1 章 jQuery 入门	3
1.1 jQuery 的优势	3
1.2 硬件和浏览器条件	4
1.3 获得 jQuery 库和 jQuery UI	4
1.4 HELLO WORLD 示例	5
1.5 本书使用的 JavaScript 规范	6
1.6 开发工具	9
1.7 调试 JavaScript 和 jQuery	16
1.8 使用 FireQuery 插件	16
1.9 小结	17
第 2 章 JavaScript 基础	19
2.1 理解数值	20
2.2 使用字符串	21
2.3 理解布尔类型	22
2.4 类型之间的比较	23
2.5 日期简介	23
2.6 其他类型	23
2.7 变量	25
2.8 理解对象	26
2.9 使用函数	29
2.10 理解执行上下文	33
2.11 作用域和闭包	33
2.12 理解访问级别	34
2.13 使用模块	35
2.14 使用 JavaScript 数组	37
2.15 扩展类型	38
2.16 JavaScript 最佳实践	39

2.17 综合示例	40
2.18 小结	40
2.19 注意	40

第 3 章 jQuery 核心技术	41
3.1 jQuery 脚本的结构	41
3.2 非侵扰式 JavaScript	52
3.3 jQuery 框架的结构	59
3.4 理解 DOM 和事件	61
3.5 与其他 JavaScript 库一起 使用 jQuery	61
3.6 小结	62
3.7 参考	62
第 4 章 选择和操作 DOM 元素	65
4.1 jQuery 选择器的功能	66
4.1.1 选择元素	66
4.1.2 CSS 样式选择器	68
4.1.3 属性选择器	70
4.1.4 位置选择器	74
4.1.5 过滤选择器	74
4.1.6 用户自定义选择器	83
4.2 遍历 DOM	83
4.3 访问并修改元素、属性和内容	90
4.4 生成 HTML	95
4.5 小结	96
第 5 章 事件处理	97
5.1 理解浏览器的事件模型	97
5.2 理解 jQuery 中的事件处理 机制	103

5.3	使用 jQuery 进行事件处理	105
5.4	使用事件	111
5.5	jQuery 新的事件 API	117
5.6	小结	120
5.7	参考	120
第 6 章	HTML 表单、数据和 Ajax	121
6.1	jQuery 数据应用程序	121
6.2	使用表单验证	123
6.3	使用 HTML 表单元素	126
6.4	Ajax 基础	130
6.5	在 jQuery 中使用 Ajax	132
6.6	小结	142
6.7	参考	142
第 7 章	动画和特效	143
7.1	为元素创建动画效果	143
7.2	用 CSS 属性创建动画	145
7.3	改变元素的尺寸	148
7.4	设计用户自定义动画	149
7.5	在 HTML5 的 canvas 元素中 创建动画	153
7.6	小结	155
7.7	参考	155

第 II 部分 jQuery 应用

第 8 章 jQuery UI 第 I 部分——

更轻松地创建 Web 界面 159

8.1	主题和样式	159
8.2	使用 ThemeRoller	161
8.3	使用 jQuery 小组件	161
8.3.1	Button	162
8.3.2	Tabs	163
8.3.3	折叠面板(Accordion)	167
8.3.4	Autocomplete	169

8.3.5	Datepicker	171
8.3.6	对话框	176
8.4	进度条	178
8.5	滑动条	179
8.6	小结	181
8.7	参考	181

第 9 章 jQuery UI 第 II 部分——

鼠标交互 183

9.1	拖曳和置放	183
9.2	排序	188
9.3	缩放元素	191
9.4	可选取元素	194
9.5	小结	196

第 10 章 编写高效的 jQuery 代码 197

10.1	优化技术	197
10.1.1	最小化 DOM 更新	198
10.1.2	更高效的循环	200
10.1.3	缓存对象	201
10.1.4	高效使用选择器	202
10.1.5	考虑完全跳过 jQuery 方法	206
10.1.6	DRY	207
10.2	使用 JavaScript 模式	209
10.2.1	使用一个单例创建一个 应用程序名称空间	209
10.2.2	Module 模式	212
10.2.3	Garber-Irish 实现	215
10.3	使用 \$.DATA()	218
10.3.1	基本的.data() API	219
10.3.2	充分利用 Data API	219
10.4	小结	222

第 11 章 jQuery 模板 223

11.1	征服字符串	223
------	-------	-----

11.1.1	分离内容与行为	225	14.2.1	单元测试的优点	288
11.1.2	代码重用	225	14.2.2	测试驱动的开发	289
11.1.3	简洁而优美	225	14.2.3	什么是一个好的 单元测试	289
11.1.4	jQuery 模板的过去、现在 和未来	225	14.3	QUnit 入门	290
11.1.5	创建 jQuery 模板	226	14.3.1	在 QUnit 中使用 equal 测试 Hello World	290
11.1.6	使用 \$.tmpl() 方法应用 模板	228	14.3.2	一个失败的 QUnit 测试	292
11.1.7	在模板中使用远程数据	231	14.3.3	使用 ok 测试真伪	292
11.1.8	模板标记	233	14.3.4	设置预期的断言数量	293
11.2	小结	243	14.3.5	其他断言	294
第 12 章	编写 jQuery 插件	245	14.3.6	测试 DOM 元素	295
12.1	插件基础	245	14.3.7	使用 noglobals 和 notrycatch	296
12.1.1	遵循 jQuery 插件的 命名规范	246	14.3.8	将测试组织为模块	298
12.1.2	如何扩展 jQuery	246	14.4	异步测试	300
12.1.3	jQuery 插件通用指南	249	14.4.1	使用 asyncTest	301
12.1.4	jQuery 插件最佳实践	254	14.4.2	模拟 Ajax 请求	302
12.2	学习和使用现有的插件 模式	262	14.5	综合示例	304
12.3	Widget Factory 概述	264	14.6	小结	307
12.4	插件开发示例	265	附录	本书中使用的插件	309
12.5	小结	268			
第 13 章	使用 jQuery Deferred 对象 进行高级异步编程	269			
13.1	\$.Deferred 基础	270			
13.1.1	Promise	270			
13.1.2	Promises/A Proposal	270			
13.2	jQuery 中的 Deferred 对象	271			
13.3	小结	286			
第 14 章	使用 QUnit 进行单元测试	287			
14.1	单元测试简介	287			
14.2	什么是单元测试	288			



第 I 部分

jQuery 基础

- 第 1 章：jQuery 入门
- 第 2 章：JavaScript 基础
- 第 3 章：jQuery 核心技术
- 第 4 章：选择和操作 DOM 元素
- 第 5 章：事件处理
- 第 6 章：HTML 表单、数据和 Ajax
- 第 7 章：动画和特效

第 1 章

jQuery 入门

本章内容

- jQuery 的优势
- jQuery 版的 Hello World 代码范例和开发工具

我们在计算机前花费的时间，也许比在其他值得关注的事情上花费的时间更多。因此对于开发人员来说，一把舒适的座椅、一台快速的电脑以及一些可以加快开发过程的软件，都是至关重要的。

本章将从基础开始，介绍 jQuery 的使用要求、获取 jQuery 的多种方法以及如何在客户端和服务端上运行 jQuery。本章还将使用 Google 的 JavaScript 标准，为本书其余章节建立一些编码规范。

本章将建立开发和调试 jQuery 和 JavaScript 代码的工作环境。在开发过程中，调试占据了主要的开发时间，因此一个优秀的调试器是非常重要的。本章将介绍 Firebug 调试器、Chrome 和 IE 调试器，包括调试 JSON 的工具、查看 HTTP headers 的工具，以及 JSLint 代码质量工具。

本章还将介绍打包.js 文件以及供产品化使用的工具、JavaScript 代码查看工具并建立一些编码规范。

1.1 jQuery 的优势

毫无疑问，jQuery 最强大的优势之一就是它消除了很多跨浏览器的兼容性问题。没有 jQuery，编写跨浏览器兼容的 JavaScript 代码是一项繁重而无趣的工作。在使用 jQuery 之后，人们不禁要惊叹 jQuery 节省了大量的时间，让代码既能在 IE 中运行，也能在 Firefox 中运行。此外，在遍历 DOM 树和选择元素方面，jQuery 的核心也非常卓越。jQuery 是非

常轻量级的，在经过最小化和压缩之后，jQuery 标准产品版只有 29kb。这是一个相当小的文件。在调试和测试时，最好使用未压缩的开发版本，它的大小为 212kb。

jQuery 的另外一个优势，是只需要相对较少的代码就可以处理相当复杂的 JavaScript 代码才能实现的功能。这是 jQuery 框架首先吸引作者的原因之一。对于想学习 Ajax“秘籍”的新手，jQuery 提供了一个较平缓的学习曲线。虽然理解 Ajax 内在的原理很重要，但是作为入门的第一步，使用 \$.ajax() 是一个不错的起点。

活跃的用户社区意味着活跃的支持能力。在本书写作时，从 Google 上搜索 jQuery 将返回大约 0.368 亿条结果。每天都有新的官方插件和第三方插件产生，它们不断地扩展 jQuery 的核心功能。很多项目都依赖于 jQuery，因此很容易找到在线的帮助(以及出版的书籍)。

但是，在某些情况下是没必要使用 jQuery 的。请记住即使 jQuery 库很小，但包含 jQuery 库具有一定的负载。如果仅仅是使用选择器根据元素的 ID 来选取一个元素，那么直接使用浏览器内建的 JavaScript 功能就足够了。但是，如果计划构建功能丰富的、动态的 Ajax Web 应用程序，那正是 jQuery 的用武之地。

1.2 硬件和浏览器条件

运行 jQuery 所需的条件很简单：一台计算机、一个智能电话或一个可以运行现代浏览器的设备。jQuery 对浏览器的要求也相当自由。官方网站列出了下列支持 jQuery 的浏览器：

- Firefox 2.0+
- Internet Explorer 6+
- Safari 3+
- Opera 10.6+
- Chrome 8+

下列浏览器存在一些已知的问题：

- Mozilla Firefox 1.0.x
- Internet Explorer 1.0–5.x
- Safari 1.0–2.0.1
- Opera 1.0–9.x
- Konqueror

在 Quirksmode 下，有很多篇幅详细地讨论了不同浏览器之间 CSS 的兼容性，包括 CSS 3。更多内容请参考 www.quirksmode.org/css/contents.html。

1.3 获得 jQuery 库和 jQuery UI

要获取 jQuery 库，除了从 jquery.com 的相关链接下载产品版和开发版的 jQuery 库之

外,还可以采用其他一些方法获取或链接到 jQuery 库。在应用程序中,可以链接到几个内容分发网络(content delivery network, CDN),比如 Google、Microsoft 和 jQuery 官方网站提到的 jQuery CDN,通过 CDN 来链接 jQuery 库。

下面的代码说明了如何从几个可用的 CDN 获取 jQuery 库。只需要简单地添加一个 `<script>` 标记,并将 `src` 属性指向想要访问的 jQuery 库的 URL 即可。



可从
Wrox.com
下载源代码

```
<script src="https://googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js">
</script> <script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.7.1.
min.js">
</script> <scriptsrc="http://code.jquery.com/jquery-1.7.1.min.js"></script>
```

代码片段 `cdns.txt`

Git repository 中还具有处于 Work in Progress(或简称为 WIP)状态的 jQuery 生成库。它是还在不断进行修正的开发版本,即不应该用于实际的产品之中。WIP 每隔一段时间就会重新生成。要使用最前卫的 WIP 版本,可以直接链接到 <http://code.jquery.com/jquerygit.js>, 或者也可以克隆 Git repository 以从头开始生成 jQuery 库。下面的代码演示了如何使用 Git 命令,克隆最新版本的 jQuery 库。

```
$ git clone https://github.com/jquery/jquery.git jquery-build
```

为了生成 jQuery 库,还需要 GNU 以便使用 `make 3.8+` 和 `Node.js .2+`。在克隆了 repo 之后,进入 jQuery 目录并执行 `make` 命令。这将生成一个最小化的 jQuery 完整版,可以通过 JSLint(后面将介绍 JSLint)运行。如果不需要这些,可以运行 `make jquery` 命令以取代 `make` 命令。

1.4 HELLO WORLD 示例

如果没有一个常见的 Hello World 程序,一本介绍程序设计的书似乎就是不完整的:



可从
Wrox.com
下载源代码

```
<html>
<head>
  <script
    src=" https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js">
  </script>
  <script >
    jQuery(document).ready(function(){
      alert('Hello World');
    });
  </script>
</head>
<body>
</body>
</html>
```

代码片段 `hello-world.txt`

简而言之，在上面的例子中，通过 Web 的在线 CDN 加载了一个最小化版本的 jQuery 库。在页面的脚本代码中，调用了 `jQuery()` 函数，并将 `document` 对象作为参数传入该 `jQuery()` 函数。`jQuery()` 函数将创建一个具有一个 `ready()` 方法的包装对象，`ready()` 方法只能接收一个函数作为参数。在浏览器中，当把页面中的 HTML 转换为文档对象模型(document object model, DOM)这一过程完成时，将调用 `ready()` 方法。此时，该函数将显示一个警告信息 Hello World。与我们看到的结果相比，在幕后还有更多的故事，如果你还不理解其中的过程，也无须担心。第 2 章将回顾 JavaScript 语言，并重点介绍 JavaScript 中管理函数的方式。经过第 2 章的学习，你就能理解在 `ready()` 方法中使用匿名函数的原因。在第 3 章和第 4 章中，将介绍 jQuery 包装器背后的相关概念，以及如何使用 jQuery 选择器从 DOM 中选择元素。经过这些学习，你将能更好地理解 `jQuery(document)` 的含义。最后，第 5 章将向你介绍 jQuery 中的事件处理，比如之前提到过的 `.ready()`。因此，尽管 Hello World 例子中的代码非常简单，但实际上涉及相当多的知识点，本书将逐一进行解答。

1.5 本书使用的 JavaScript 规范

本书将坚持使用 Google JavaScript Style Guide 的部分子集和 jQuery Core Style Guide lines 作为编码规范，前者可以在网址 <http://google-styleguide.com/svn/trunk/javascriptguide.xml> 找到，后者可以在网址 http://docs.jquery.com/jQuery_Core_Style_Guidelines 找到。Google JavaScript Style Guide 定义了下列规范：

- 除了意图明确地要创建一个全局变量之外，总是使用 `var` 声明变量
- 总是使用分号。这对于压缩代码非常重要
- 常量使用大写，每个单词之间用一个下划线分隔
- 函数、变量和方法名都采用驼峰拼写法，并且将整个变量名的第一个字母小写
- 类和枚举名也采用驼峰拼写法，但整个变量名第一个字母大写

当用例子演示不遵循这些规则的情况时，可以看到这些规则的例外情况。

jQuery 团队还发表了一些开发 jQuery 核心库的规范。在该规范中描述了下列规则：

- **使代码有间隔：**在代码中明确地使用空格，并使用制表符(tab)缩进代码。在代码行末尾不要使用空格字符，在空行中也不应该使用空格字符。下面的代码描述了一种在代码中使用间隔的较好方式：

```
if ( test === "test string" ) {  
    methodCall( "see", "our", "spacing" );  
}
```

- **使用注释：**对于多行注释使用 `/* */`，对于单行注释，使用 `//`，在注释上保留一个空行。请将单行注释放在所注释代码行之上，并且在注释行中仅包含注释的说明。

```
// 注释
var x = 'blah';

var x = 'blah'; // 不良方式
```

- **相等性：**总是使用等同(===)进行比较，而不是简单地使用等于(==)进行比较。对这一规则，jQuery 团队的一个例外，就是在检查 null 值时使用简单的等于操作符进行比较。正如规范所说，执行 == null 或者 != null 的比较操作实际上非常有用，因为当值为 null 或 undefined 时，这种比较会成功(或失败)。
- **以块方式组织代码：**对于控制结构(if/else/for/while/try)总是使用花括号，并且总是以多行的代码块方式来编写代码。不要使用没有花括号的单行 if 语句。应该将起始花括号与 else/else if/catch 放在同一行上。建议不要使用三元操作符来取代 if/else 语句。下面是一些例子：

```
//不良方式
if(stuffHappens ) alert('blaaaah');

//良好方式
if(stuffHappens ) {
    alert('blaaaah');
}

//良好方式
if( option ) {
    //代码
} else {
    //代码
}
```

- **函数调用格式：**在函数调用参数的两侧包含一个多余的空格。但当函数调用是嵌套的、或是函数的参数为空、或者传入的参数是一个对象字面量或数组时例外。

```
//正确的函数调用方式
f(arg );
f( g(arg) );
f();
f({ });
f([ ]);
```

- **数组和对象：**对于空对象或数组字面量不要使用额外的空格，但在逗号和冒号之后使用一个空格：

```
var a = {};
var b = [];
var c = [ 1, 2, 3 ];
```


- **对变量/对象赋值：**在赋值语句后总是使用一个分号结尾并结束该行。Google Style Guide 也要求，在一行代码结尾总是使用分号。
- **类型检查：**表 1-1 描述了执行类型检查的代码。

表 1-1 类型检查

对 象	类型检查表达式
String	<code>typeof object === "string"</code>
Number	<code>typeof object === "number"</code>
Boolean	<code>typeof object === "boolean"</code>
Object	<code>typeof object === "object"</code>
Element	<code>object.nodeType</code>
null	<code>object === null</code>
null 或 undefined	<code>object == null</code>

表 1-2 列出使用 jQuery API 对对象执行类型检查的方法。

表 1-2 使用 jQuery API 执行类型检查

对 象	用于类型检查的 JQUERY 方法
Plain Object	<code>jQuery.isPlainObject(object)</code>
Function	<code>jQuery.isFunction(object)</code>
Array	<code>jQuery.isArray(object)</code>

表 1-3 列出了用于对 undefined 执行类型检查的方法。

表 1-3 检查 undefined

对 象	
Global Variables	<code>typeof variable === "undefined"</code>
Local Variables	<code>variable === undefined</code>
Properties	<code>object.prop === undefined</code>

- **RegExp：**在 `.text()` 或 `.exec()` 中创建正则表达式(RegExp)
- **字符串：**使用双引号而不是单引号

开发规范的文档中还提到了使用 JSLint 进行验证，JSLint 将在下一小节“开发工具”中进行介绍。

Google JavaScript Style Guide 与 jQuery 的 Style Guide 之间也存在着一些差别。例如，Google JavaScript Style Guide 建议使用单引号，而 jQuery 团队则将使用双引号作为一种标准。在本书中我们将遵循 jQuery 团队的建议。

1.6 开发工具

在使用 JavaScript 和 jQuery 开发时,除了传统的编辑器/浏览器之外,还有多种工具可供选用。如果你更喜欢面向命令行方式的开发风格,则可以使用 jconsole 和 Rhino。jconsole 是一个 Web 应用程序,它允许用户以交互方式输入 JavaScript 代码。jconsole 还提供了一些函数供用户使用,比如 print()、load()和 clear()。在每行代码之后按下 Ctrl+Enter 组合键,就可以在 jconsole 中输入多行代码。只需要按下 Enter 键就可以执行当前文本输入区域中的代码。jconsole 还维护着一个你所输入命令的历史纪录,按上下箭头键就可以访问或使用前一个或后一个输入的命令。图 1-1 显示了 jconsole 的使用界面。

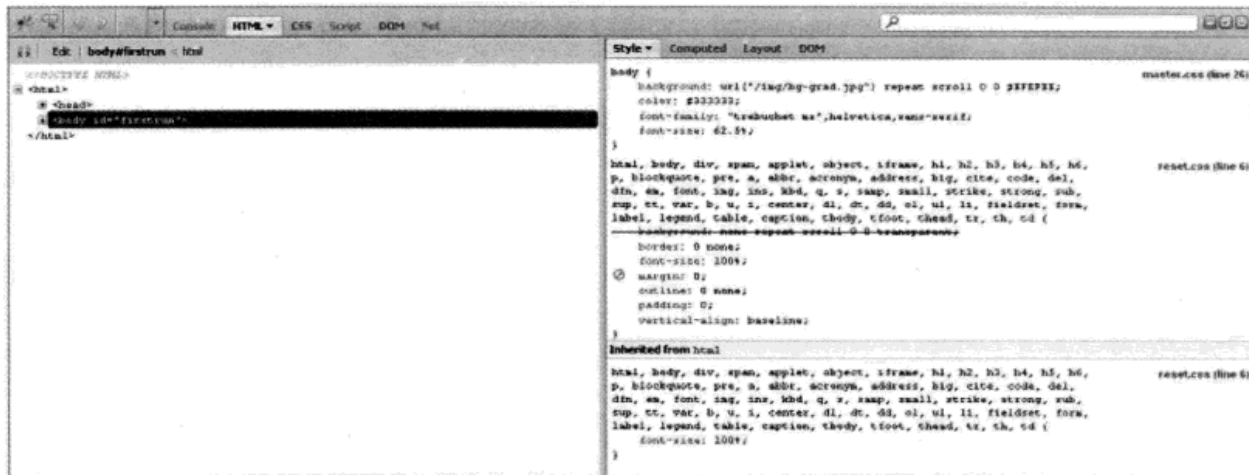


图 1-1

在 jconsole 中要加载 jQuery,请使用 load()函数并传入一个 jQuery 库的 CDN URL 作为参数。例如:

```
load('https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js');
$(document).click(function(){ alert('It worked!'); });
```

jconsole 还具有按 tab 键自动补全输入的功能,通过输入 scope()函数,还可以动态改变当前的执行上下文。对于快速地测试小段的 JavaScript 代码, jconsole 确实是一个有价值的工具。

可以在某个浏览器环境中使用本书来学习 jQuery,但在服务器端也可以使用 jQuery。Mozilla Rhino 是一个用 Java 实现的 JavaScript 解释器。它通常用于在 Java 应用程序中嵌入 JavaScript。它提供了 JavaScript shell、调试器和编译器的功能。此外还有 Envjs 框架,它是由 jQuery 的创建者 John Resig 编写的。在服务器环境中也可以使用 jQuery。Envjs 是一个可移植的 headless browser 的 JavaScript 实现,或者说缺少用户界面但提供了一个脚本编程环境的浏览器。在其他宿主环境(host environment)中也可以获得相同的效果,比如 Google V8,但作者将使用 Rhino 作为例子。

表 1-4 列出了获得和运行 jQuery 和 Rhino 的条件。

表 1-4 在 Rhino 中运行 jQuery 的条件

DEPENDENCY	功 能
Rhino	命令行模式的 JavaScript 解释器
JLine(optional, but useful)	Java 控制台输入库、类似于读取行(readline-like)的功能
Envjs	headless browser、脚本编程环境

要获得与读取行类似的功能，请在运行 Rhino 时带上 JLine 库。这样一来，就可以通过按上箭头键访问历史命令，并使用左箭头键和右箭头键来编辑字符。下面的代码块说明了如何启动 JLine。

```
java -cpjs.jar:jline.jarjline.ConsoleRunner \
org.mozilla.javascript.tools.shell.Main -opt -1
```

下面的小节演示了如何在 Rhino 中运行 jQuery。具有提示符 js>的行是输入行。

```
js> load('env.rhino.1.2.js');
[ Envjs/1.6 (Rhino; U; Linux i386 2.6.24-25-generic; en-US; rv:1.7.0.rc2)
Resig/20070309
PilotFish/1.2.13 ]
js>window.location = "http://localhost"
http://localhost
js> load('jquery.js');
js>jQuery

function (selector, context) {
    return new jQuery.fn.init(selector, context);
}
```

对于正规的浏览器，Firefox 也提供了一个简化开发人员工作的插件。其中最流行的一个插件是 Firebug，它是一个 Web 开发的调试工具。Firebug 允许实时编辑 HTML 和 CSS、调试并监控 Web 应用程序的运行。图 1-2 显示了 Firebug 的界面。

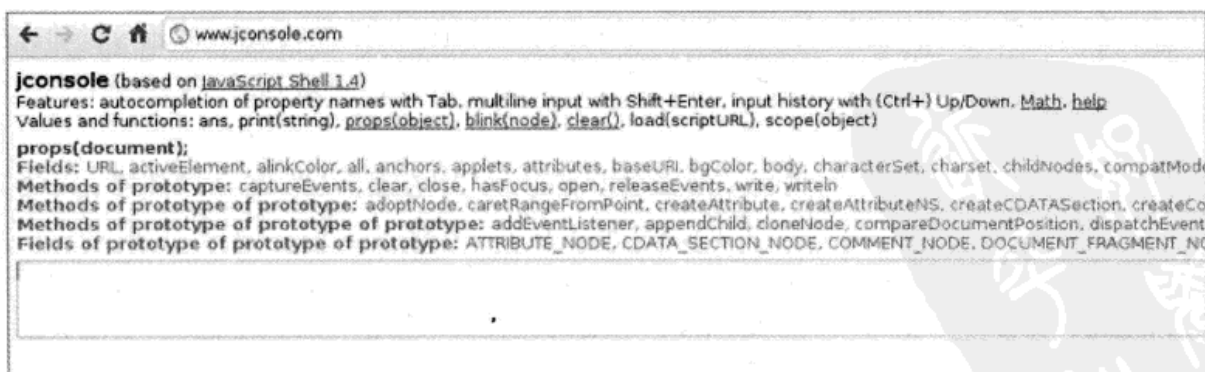


图 1-2

要安装 Firebug，只需要在菜单栏上点击 Tools 下的 Add Ons 子菜单，在打开的对话框中点击 Get Add Ons，然后搜索 Firebug 即可。找到 Firebug 插件后，点击 Add to Firefox。

在安装完 Firebug 插件之后，在浏览器窗口右下角可以看到一个昆虫图标。点击该图标将打开 Firebug。

Firebug 包含 6 个选项卡：Console、HTML、CSS、Script、DOM 和 .NET。每一个选项卡都包含了一些简化 Web 开发人员工作的特性。

Console 选项卡提供了一个输出界面，用于观察警告和错误消息、日志、调试信息、普通信息、XmlHttpRequest 请求和响应对象。

Firebug 把一个名为 console 的对象加载到全局名称空间中。使用该对象可以将应用程序内部工作的消息输出到控制台。console 对象包含了一些方法，可以从 http://getfirebug.com/wiki/index.php/Console_API 上查阅 console 对象的方法。表 1-5 列出了 console 对象包含的方法的一个简要子集，这些方法的描述来自于 Firebug Wiki。

表 1-5 Console 对象的一些方法

方 法	描 述
<code>console.log(object[, object, ...])</code>	向控制台输出一条消息
<code>console.debug(object[, object,...])</code>	向控制台输出一条消息，包括一个到该 <code>console.debug</code> 代码行的超链接
<code>console.info(object[, object,...])</code>	向控制台输出一条消息，包含可视的“信息”图标和着色的代码，以及一个到该 <code>console.info</code> 代码行的超链接
<code>console.warn(object[, object,...])</code>	向控制台输出一条消息，包含可视的“警告”图标和着色的代码，以及一个到该 <code>console.info</code> 代码行的超链接
<code>console.error(object[, object,...])</code>	向控制台输出一条消息，包含可视的“错误”图标和着色的代码，以及一个到该 <code>console.info</code> 代码行的超链接
<code>console.assert(expression[,object,...])</code>	测试一个表达式是否为 true。如果不为 true，该方法向控制台输出一条消息并抛出一个异常
<code>console.clear()</code>	清除控制台信息

profiler 用于应用程序的性能分析。利用 profiler 分析性能非常简单，只须点击一下 Profile 按钮即可。加载要分析的页面，点击 Console 选项卡之下的 Profile 按钮，使用应用程序一段时间后，再次点击 Profile 按钮，就会生成一个性能报告。

HTML 选项卡允许你在“运行时”查看 DOM 和编辑 HTML 代码。它以一个树型部件(widget)的形式来显示 HTML 代码的标记结构，展开其中的节点就可以显示每一个节点的子节点。在 CSS 选项卡中，可以编辑样式和布局，并可以整洁地显示出每一个级联样式的样式定义。

Firebug 中最有用的功能，也许是交互式的命令行，它可以在正在开发的 Web 页面的上下文中执行 JavaScript 的代码。不必手工加载 jQuery，也不依赖于其他设置。控制台还具有命令补全功能(command completion)。在 Script 选项卡中，可以添加或移除断点、随时查看跟踪栈、查看变量的值。

在其他浏览器中, 也通过 Firebug Lite 来使用 Firebug。要使用 Firebug Lite, 只需要在 HTML 文件的<script>标记中包含在线 firebug-lite.js 文件的链接:

```
<script src="https://getfirebug.com/firebug-lite.js"></script>
```

除了 Firebug 之外, 另外一个 Firefox 插件 Live HTTP Headers 可以为数据流和应用程序提供一个详细的视图。Live 在这里的含义是, 该 HTTP 的 header 信息是实时显示的。

要安装 Live HTTP Headers 插件, 只须采用与安装 Firebug 插件类似的过程即可。在安装完 Live HTTP Headers 插件之后, 可以从 Tools 下的 Live HTTP Headers 子菜单来访问 Live HTTP Headers。它将打开这个对话框, 显示 http 请求的所有有用的调试信息, 比如 server type、encoding type、cookies、host 等。replay 特性允许你编辑 http 请求的 header 信息, 并重新发送一个 URL, 这对调试 Ajax 非常有用。

当试图在 Firefox 中直接加载 JSON 数据时, Firefox 浏览器默认的行为是为用户提供一个 Download File 对话框。使用 JSONView 扩展(extension), Firefox 将以一种便于阅读的方式显示 JSON 数据。此时 JSON 中的节点是可压缩或展开的, 与 Firefox 直接解释 XML 数据时的显式方式类似。例如, 对于下面的.json 文件:

```
{
  "users": {
    "user": [{
      "name": "Tom",
      "email": "tom@bigcorp.com",
      "role": "admin"
    },
    {
      "name": "Nick",
      "email": "nick@bigcorp.com",
      "role": "employee"
    },
    {
      "name": "Lynn",
      "email": "lynn@bigcorp.com",
      "role": "manager"
    },
    {
      "name": "Carol",
      "email": "carol@bigcorp.com",
      "role": "manager"
    }
  ]
}
```

在 JSONView 中, 以上 JSON 数据将输出如下所示的代码片段。减号指出这是一个已展开的节点, 加号则表示这是一个可展开的节点。最后两条记录是收缩的。

```
{
```

```

- users: {
  - user: [
    - {
      name: "Tom",
      email: "tom@bigcorp.com",
      role: "admin"
    },
    - {
      name: "Nick",
      email: "nick@bigcorp.com",
      role: "admin"
    },
    + { ... },
    + { ... }
  ]
}
} name: "Lynn" email: "lynn@bigcorp.com"
role: "manager"

```

Google Chrome 也提供了一个顶尖的开发环境。Chrome 浏览器默认带有一组开发者工具。Google Chrome 还在积极地开发这些工具，因此在 Chrome 浏览器中将不断包含新的特性和改进。Chrome Developer Tools 也具有 Firebug 的全部优点：一个交互式的命令行、元素查看器、添加或移除断点的功能等。要访问 Chrome Developer Tools，只须按下 Shift+Ctrl+I 组合键。

图 1-3 显示了 Chrome Developer Tools。

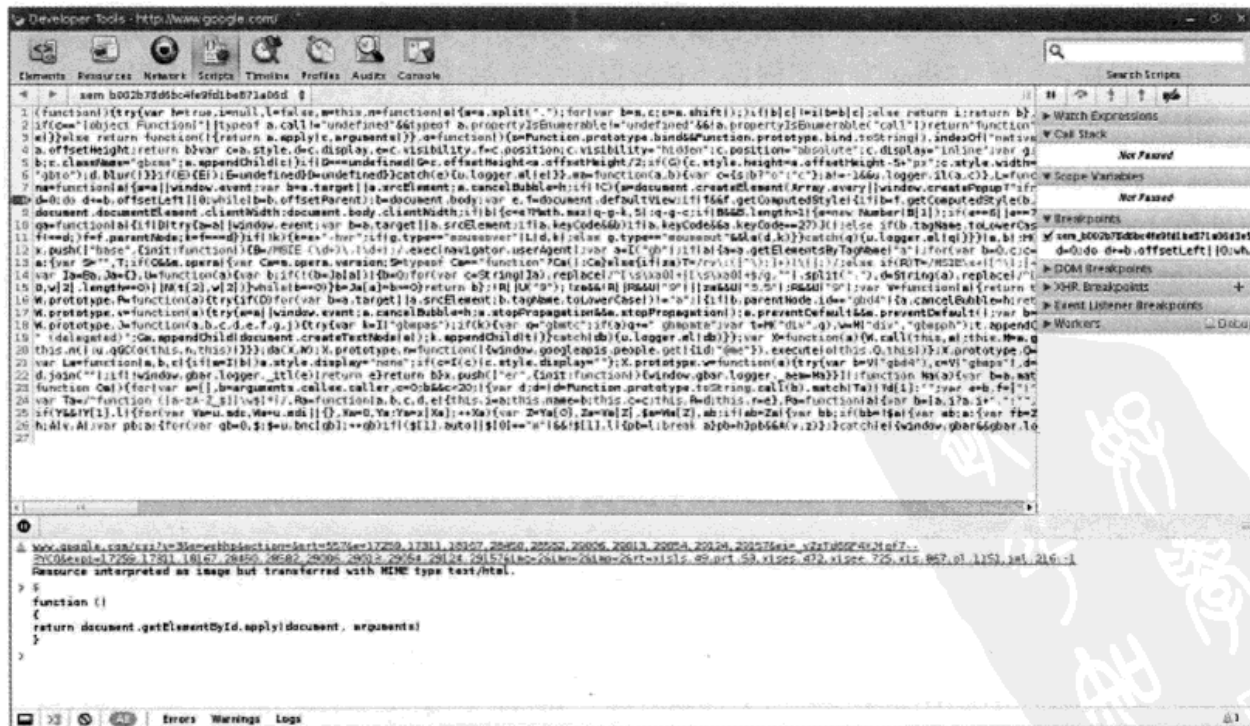


图 1-3

Microsoft 永远都不甘落后，它在 Internet Explorer 中包含了一组自己的开发者工具，但不像 Firefox 的插件或扩展那样完美。人们称之为 F12 Developer Tools，这是一个方便的提醒，只须按下 F12 键就可以访问 F12 Developer Tools。F12 Developer Tools 与 Chrome 的 Inspector 和 Firefox 的 Firebug 非常类似。图 1-4 显示了 Internet Explorer 的 F12 Developer Tools。

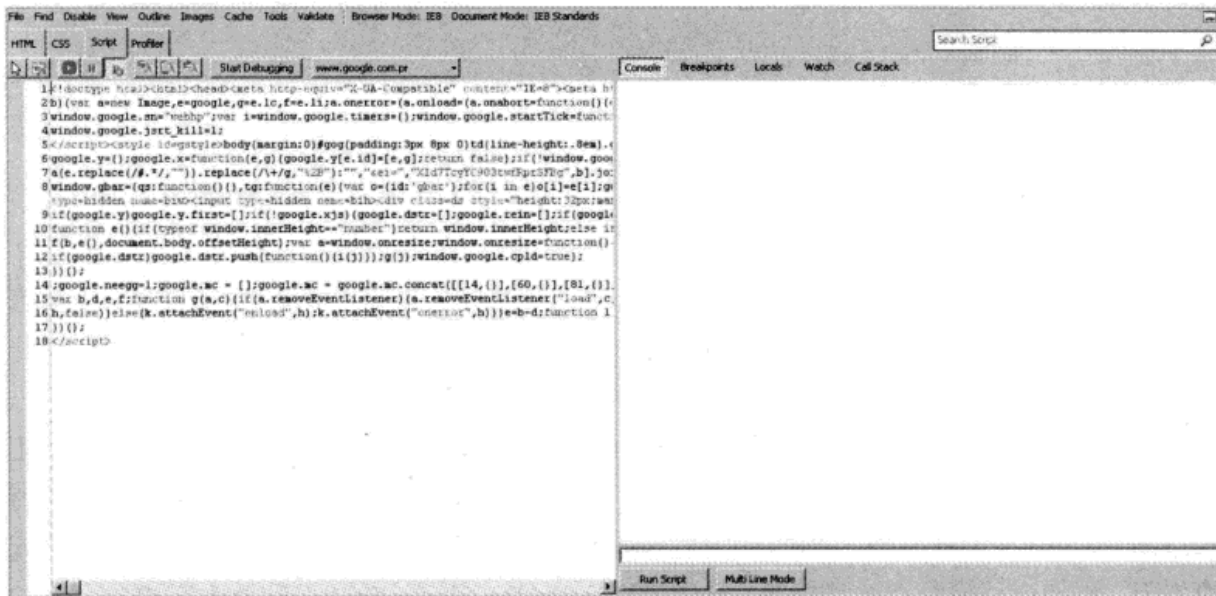


图 1-4

另外一个有用的工具是 JSLint，即 JavaScript 代码质量工具，它的功能是检查并验证 JavaScript 代码。Douglas Crockford 对它做了最好的解释：“JSLint 接收 JavaScript 源代码并对其进行扫描。如果 JSLint 发现了问题，它就返回一个描述该问题和它在源代码中相应位置的消息”。与 jconsole 类似，JSLint 提供了一个方便的 Web 应用程序来验证你的 JavaScript 代码，要使用 JSLint 请访问 www.jslint.com。另外，也可以从 Github 克隆 JSLint 的源代码，地址为 <https://github.com/douglascrockford/JS�int.git>。在其源代码中，可以找到一个名为 fulljslint.js 的文件，可以在 Web 浏览器之外运行该文件。使用 Mozilla Rhino，只需要使用一个简单的脚本就可以通过命令行来检验 JavaScript 或 jQuery 代码。

```
load('JSLint/fulljslint.js');
varsrc = readFile(arguments[0]);
var passes = JSLINT(src);
print(passes ? 'Passes Lint Validation' : 'Failed Validation');
```

在上面的代码中，load()、readFile()和 print()都是 Rhino 特有的函数。load()函数接收一个指向 fulljslint.js 文件的输入字符串作为参数，它将 fulljslint.js 代码加载到内存中并执行代码，以允许你访问 JSLint()函数。readFile()函数以参数数组的形式，获取以命令行方式传入的第一个参数，并将源文件读入一个字符串中。JSLINT()函数将对变量 src 的内容进行验证，它返回一个 Boolean 对象，指出该字符串的验证是否有效。最后，使用一个三元操作符根据读取的文件是否通过验证向控制台输出相应的信息。打开一个终端，尝试对一个

简单 JavaScript 文件进行验证:

```
$ rhino runLint.js test.js  
Passes Lint Validation
```

如果觉得上面的方式有点繁琐,还可以使用另外一个项目 `jslint4java`,地址是 <http://code.google.com/p/jslint4java/>,它将所需的命令简化为一行代码,比如下面的例子:

```
$ java -jar jslint4java-1.4.jar test.js
```

除了 JSLint 之外,还有一些其他的代码质量工具日渐流行,比如 JSHint(www.jshint.com/)。它最初是从 JSLint 中分化而来,目的是创建一个可配置性更强的 JSLint 版本,它不强制使用某一种特定的编码风格。JSHint 已经成长为一个独立的项目,它具有自己的目标和理念。如果你不喜欢 JSLint 中特定的、不可配置的规则,或者尝试一下新方法,JSHint 是一个很好的选择。

最后,在任何模块或插件的开发完成后以及用 JSLint 验证之后,开发人员可能希望将 .js 文件最小化以作为产品使用。目前已经有多种可供选择的 JavaScript 文件最小化工具,但作者在这里只简单介绍其中几种。一些开源的 JavaScript 文件压缩工具包括:YUI Compressor、Google Closure Compiler、UglifyJS 和 JSMIn。YUI Compressor 在将 JS 最小化的同时,还提供了将 CSS 最小化的功能。尽管 YUI Compressor 这种一站式满足所有最小化需求的工具很便利,但目前占主导地位的工具是 UglifyJS——一个基于 NodeJS 的工具、Google 的 Closure Compiler。二者都可以用在 JavaScript 性能要求较高的项目中。UglifyJS 是 jQuery 项目自身选用的最小化工具,基于 Java 的 Closure Compiler 则是 HTML5 Boilerplate(<http://html5boilerplate.com>)选用的最小化工具。二者都可以从命令行运行,这允许你在运行 JSLint 的时候自动执行最小化的处理。HTML5 Boilerplate 项目提供了一个功能完备的 Ant 构建脚本,它既执行最小化,也执行 JSLint 或 JSHint(以及更多功能)。

如果想亲自动手实践一下,下面的代码示例说明,从命令行运行 UglifyJS 和 Closure Compiler 是多么简单。

假设 NodeJS 正在运行,那么安装 UglifyJS 非常简单,只需要使用下面的代码:

```
npm install -g uglify-js
```

要运行 UglifyJS 也很简单,只须调用 UglifyJS,设置 -o(输出)标记,并设置一个输出文件名,最后再传入要最小化的输入文件名即可,比如下面的代码:

```
uglifyjs -o app.min.js app.js
```

使用 Closure Compiler 也非常简单。只需要从 <http://closurecompiler.googlecode.com/files/compiler-latest.zip> 下载最新版本的 jar 文件,将其解压后从命令行运行下面的代码。它采用与 UglifyJS 类似的模式,需要提供一个输入文件和一个输出文件作为命令行的参数。

```
java -jar compiler.jar --js app.js --js_output_file app.min.js
```


如果不是这方面的高手，或者不知道如何使用命令行，那么也不必担心。采用一些基于 Web 的应用程序也可以获得相同的效果。

1.7 调试 JavaScript 和 jQuery

接下来将介绍如何使用本章所讨论的工具来捕捉 JavaScript 代码中的错误，特别是 Firebug(或 Chrome Developer Tools)和 JSLint。与其使用 alert 方法来显示变量或对象信息，不如设置一个断点，以便查看脚本所操作的对象或变量的值。另外，还可以自由地使用实时编辑功能。

现代 Web 开发的方式是将内容(标记代码)与行为(功能)分离，这样做有很多原因。其中一个原因，就是为了将 JavaScript/jQuery 代码分离出来，放在一个外部的.js 文件中，这有助于调试器正确地匹配引发错误的实际代码行编号。此外，这样做的好处是实现非侵扰式的 JavaScript 编程。

用测试驱动方式来开发项目。本书第 14 章将介绍如何使用 QUnit 在 JavaScript 项目中添加单元测试。你可能已经知道，尽管 jQuery 解决绝大多数跨浏览器的问题，但依然需要在不同浏览器和环境中测试我们所编写的代码，包括在类似于 iPhone 这样的移动设备上。使用功能全面的测试工具，可以更容易地检查和发现浏览器的 bug。

对于小段代码，比如一行代码，可以使用控制台应用程序来测试是否可以获得我们所期望的结果，比如 jconsole 和 Rhino。

确保显而易见的代码是正确的。哪怕是最有经验的程序员也可能会漏掉一些显而易见的东西。比如在包含 jQuery 库时没有添加<script>标记，过了很长时间才发现，或者惊愕地发现路径中有错误的输入。如果\$变量未定义，调试器将很快提示你这一问题。

注意特定于浏览器的问题以及浏览器的特殊行为，这是非常重要的。在 Firefox 或 Safari 浏览器中进行开发时，在对象字面量中使用结尾的逗号非常重要，因为这些脚本项目可能会在 IE 浏览器中使用。

在本书中笔者将反复强调一点，真正理解 JavaScript 是很有必要的。jQuery 的学习曲线较平缓，对于初学者来说，jQuery 是一个美妙的框架，可以创造出令人炫目的 JavaScript 效果。然而，从事全面开发的高级用户不应该将 jQuery 视为一个黑箱。jQuery 拥抱并扩展了 JavaScript，因此高级开发人员应该对 JavaScript 有充分的理解。

此外还有很多其他常见的小技巧。比如确保 HTML 和 XML 是格式良好的、使用诸如 Mercurial 或 Git 之类的版本控制工具、利用 Google 查找解决问题的办法、或者在遇到难题时看看有没有堆栈溢出问题。

1.8 使用 FireQuery 插件

在 jQuery 宝库中还有另外一个 Firefox 插件：FireQuery 插件，它增强了 Firebug 以便

更好地处理 jQuery。它的特性包括:

- 在控制台中，可以查看 jQuery 在运行时动态添加的元素和处理程序
- 在 mouseover 事件中，将 jQuery “集合”中的成员元素在窗口中高亮显示
- 在 Firebug 控制台中识别并呈现特定于 jQuery 的表达式

FireQuery 插件最有趣的功能之一，就是它的 jQueryify 按钮，它可以动态地将 jQuery 加载到浏览器查看的任何 Web 页面。如果你正使用 jconsole 进行测试，这一功能将非常有用，因为它消除了在使用控制台使用 load() 命令从某个 CDN 加载 jQuery 的步骤。

在安装好 FireQuery 之后，可以通过访问网址为 <http://firequery.binaryage.com/test/index.html> 的测试页面，检查 FireQuery 的各项功能是否能正常工作。图 1-5 显示了成功安装 FireQuery 之后的情形。

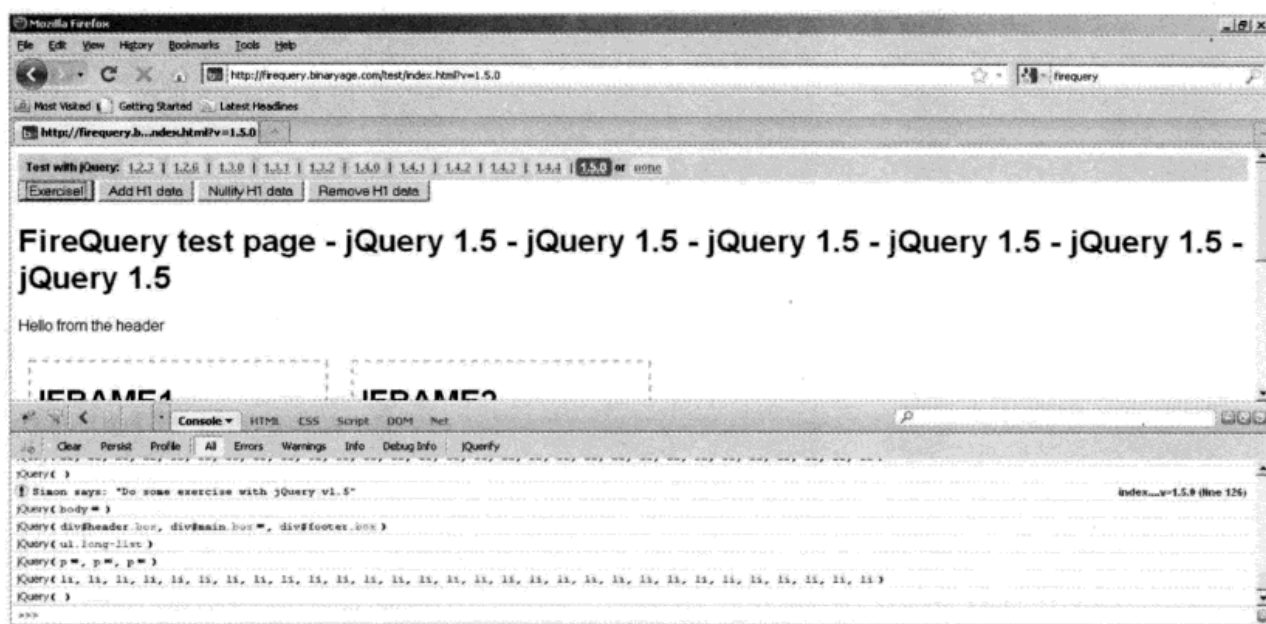
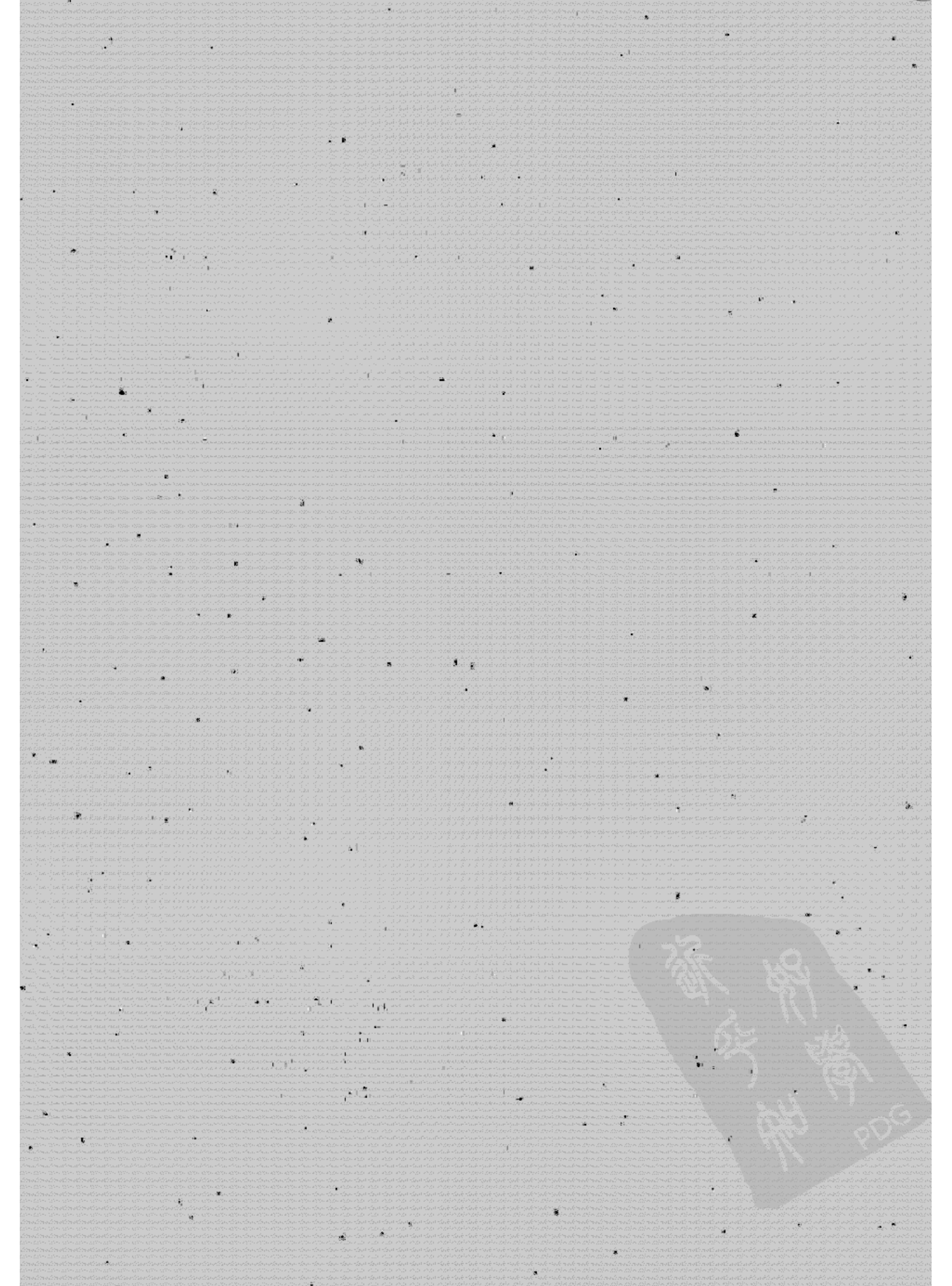


图 1-5

1.9 小结

本章介绍了获取、构建和运行 jQuery 的多种方法。不仅如此，本章还介绍了如何从服务器端运行 JavaScript 代码，而不仅仅是从传统的客户端浏览器运行 JavaScript。在学习完本书后面开发 jQuery 插件的章节后，可以使用本章介绍的这些技术，将你的代码最小化以便部署给更多的用户。此外，本章还简要地介绍了一些调试工具。

在后面的章节中，将着重介绍如何编写 jQuery 代码并充分利用本章介绍的这些工具。在介绍 jQuery 代码之前，第 2 章将简要地介绍一下 JavaScript。



第 2 章

JavaScript 基础

本章内容

- 理解 JavaScript 的基础知识
- 使用变量、函数和对象
- JavaScript 的作用域、闭包和执行上下文

人们很容易忘记 jQuery 是一个 JavaScript 框架，在使用 jQuery 一段时间之后，人们可能会感觉 jQuery 就像是一门独立的语言一样。在 jQuery 源代码的内部包含了丰富的示例，这些例子演示了如何使用大量优秀的 JavaScript 技术。要想真正精通 jQuery，必须首先精通 JavaScript。JavaScript 是一种非常强大和精深的语言。所有主流浏览器都支持 JavaScript 语言，在使用了 Ajax 的现代 Web 应用程序中，JavaScript 是真正的驱动力。在移动 Web 平台中，JavaScript 也崭露头角。

JavaScript 继承了 Java 语言中的一些名称和命名规范，但它从 Java 语言继承的也就仅仅是命名规范而已。与 Java 相比，JavaScript 是一种动态类型的、面向对象的脚本语言，它运行在宿主(host)环境之中，它的宿主环境通常是一个 Web 浏览器。JavaScript 中没有类的概念，它使用原型(prototype)实现继承。JavaScript 还具有函数式程序设计语言的特性：闭包、将函数作为第一类(first-class)对象、匿名函数、高阶函数等等。实际上，JavaScript 更接近于 Lisp 语言，而不是 Java 语言。

本章复习了理解 jQuery 工作原理所需要的 JavaScript 语法子集，介绍了 JavaScript 的核心概念以及如何在项目中更好地应用 JavaScript。本章首先简要地介绍了 JavaScript 语言的数据类型和面向对象程序设计，然后深入分析了 JavaScript 的函数式程序设计特性。在本章末尾列出了使用 JavaScript 语言时应该注意的一些事项。无论何时，当你对 JavaScript 语言的工作原理感到困惑时，都可以把本章的内容作为一个参考。另外，在运行本章中的示例时，可以使用 jconsole 或者 Rhino 进行测试。

2.1 理解数值

与其他任何程序设计语言一样, JavaScript 可以处理诸如数据或文本的值。一门语言可以使用的值的类型, 称为该语言的数据类型。JavaScript 支持基本的数值和字符串的数据类型。在 JavaScript 中, 所有数值都是 64 位双精度的, 取值范围从 $-5e-324$ 到 $1.7976931348623157e308$ 。也就是说, 在 JavaScript 中整数和浮点数之间并没有什么区别, 二者都是数值。下面的例子使用了 `typeof` 操作符进行演示:



可从
Wrox.com
下载源代码

```
> typeof 1;
number
> typeof 1.5;
number
```

代码片段 `typeof.txt`

所有 JavaScript 数值都按照 IEEE-754 双精度二进制数标准进行表示。当执行算术运算时应该注意一些问题。例如, 在把两个数值相加时, 在你的脑海中这是一个通用的操作, 然而在 JavaScript 中可能会获得令人大感意外的结果, 下面的代码演示了这一问题:



可从
Wrox.com
下载源代码

```
> .1 + .2
0.30000000000000004
```

代码片段 `unexpected-addition.txt`

JavaScript 没有内置的十进制数据类型, 但 JavaScript 为数值提供了两个方法: `toFixed` 和 `toPrecision`, 这两个方法可以按照固定位数的小数来格式化数值。下面的代码演示了这两个方法的使用:

```
> var num = 1234.12345123;
> num.toFixed(2);
1234.12

var num2 = 1234.12345123 ;
num2.toPrecision(8);
1234.1234
```

代码片段 `decimal.txt`

如果使用了一个超出 64 位范围的数值, 或者获得一个超出 64 位范围的值, JavaScript 将返回一个特殊数值: `Infinity`(无穷大)或者 `-Infinity`(负无穷大)。除数为 0 将返回 `Infinity`。其他特殊值还包括 `NaN`, 它表示一个“非数值”, 它是一个容易产生错误的值, 常常是一

些 bug 的根源。

当试图将一个无效字符串对象转换为一个数值时, 结果为 NaN 值。NaN 具有“毒性”, 在 NaN 值与数值之间执行一个操作将返回一个 NaN 值。可以使用内置的 isNaN() 函数来检查一个变量是否是 NaN 值。



可从
Wrox.com
下载源代码

```
> 10*1+100 - 1 - NaN
NaN
>var x = NaN;
>isNaN(x);
true
```

代码片段 isNaN.txt

JavaScript 支持八进制(基数为 8)和十六进制(基数为 16)。八进制字面值用一个 0(即零)作为前缀, 十六进制数值则以一个 x 作为前缀。

JavaScript 内置的 Math 对象用于常见的数学运算。例如, 可以使用 Math.round() 方法获得两位数的精度。



可从
Wrox.com
下载源代码

```
Math.round((.1+.2)*100)/100
0.3
```

代码片段 math-round.txt

充分利用内置对象可以节省时间、提高效率。

2.2 使用字符串

字符串是一个由 0 个或多个 16 位 unicode 字符组成的系列, 使用单引号或双引号将字符串括起。这里强调它是 unicode 字符, 是出于在国际化环境中使用 JavaScript 的重要性。JavaScript 中没有为字符定义特殊的数据类型。字符串也是(不可变的)对象, 因此字符串具有一些相应的属性和方法。



可从
Wrox.com
下载源代码

```
>"Test String".indexOf("S")
5
>"Test String".charAt(5)
S
```

代码片段 stringMethods.txt

在后面的内容中, 将介绍如何扩展内置的 String 对象以满足开发人员的需要。

2.3 理解布尔类型

Boolean 类型表示 true 值和 false 值。在适当的上下文中，比如在一个 if 语句中，任何条件判断的值都将被转换为 Boolean 值以判断“真”或“假”。在判断条件中，空字符串、NaN 值、null、undefined、数值 0(即零)和关键字 false 都将被计算为 false，其他的任何值都将被解析为 true。



可从
Wrox.com
下载源代码

```
if(''){
    console.log('something happens');
} else {
    console.log('nothing happens');
}
nothing happens
```

代码片段 basic-boolean.txt

JavaScript 支持的布尔操作包括：逻辑与(&&)、逻辑或(||)和逻辑非(!)。在很多常见任务中，布尔操作对于检验要求输入的字段非常有用。



可从
Wrox.com
下载源代码

```
function validate(){
    var name_input = 'Jimmy';
    var age_input;
    return name_input && age_input;
}

if(validate()){
    console.log('pass');
} else {
    console.log('fail');
}

//输出 fail
```

代码片段 boolean-operations.txt

NaN 值表示一个非数值的值。请输入下面的代码：



可从
Wrox.com
下载源代码

```
>typeof NaN
number
```

代码片段 typeofNaN.txt

结果很奇怪吧！？这是 typeof 操作符奇怪的行为之一。

2.4 类型之间的比较

JavaScript 具有等于(==)操作符和等同(===)操作符。==操作符是危险的，因为它在执行比较之前，强制执行类型转换。例如：



可从
Wrox.com
下载源代码

```
> 1 == "1";
true
```

代码片段 *simple-comparison.txt*

显然，这不是我们想要的比较结果。如果左操作数和右操作符真正地完全相同，===操作符才会返回 true。



可从
Wrox.com
下载源代码

```
> 1 === "1";
false
```

代码片段 *strict-comparison.txt*

对应地还有!=和!==操作符。请总是使用===和!==操作符。

2.5 日期简介

JavaScript 内置了 Date 对象，可以使用 new 操作符和 Date()构造函数来创建 Date 对象(后面将介绍原型和构造函数)，Date 对象用于表示日期和时间。不带任何参数创建一个新的 Date 对象，获得的是一个表示当前日期和时间的 Date 对象。



可从
Wrox.com
下载源代码

```
> var thisMoment = new Date();
> console.log(thisMoment);
Sun Jan 30 2011 21:37:19 GMT-0400 (AST)

> thisMoment.getFullYear();
2011
```

代码片段 *date.txt*

虽然 Date 是一个方便的对象，了解该对象当然还是有用的。强烈建议使用开源的 Date.js 库来执行日期/时间的计算，读者可以从 www.datejs.com 找到该 js 库。

2.6 其他类型

声明一个变量时未对其赋值，或者访问了一个不存在的对象属性，结果都会产生一个

称为 `undefined` 的类型。`null` 是 JavaScript 的一个内置对象，它表示没有值。在执行比较操作时，`undefined` 和 `null` 二者都被转换为 `false` 值，但是最好避免使用 `undefined`。在很多 JavaScript 解释器中，`undefined` 是可以重新赋值的，因而可能会产生存在弊病的代码：



可从
Wrox.com
下载源代码

```
undefined = true;
if (undefined){
    console.log('tricked you!');
}
```

//输出: tricked you!

代码片段 `undefined-equals-true.txt`

要全面了解 `null` 与 `undefined` 的差异，请参考 www.gibdon.com/2006/05/javascript-difference-between-null-and-undefined.html 页面的解释。

下面列出了 JavaScript 支持的各种数据类型。正则表达式，或者称为 RegEx 不在本书的介绍范围。

- Number
- String
- Boolean
- Object
- Function
- Array
- Date
- RegEx
- Null
- Undefined

在使用 `try/catch` 语句时，某些附加的内置 `error` 类型非常有用。通常在 `throw` 语句中创建 `error` 对象。



可从
Wrox.com
下载源代码

```
try{
    throw new Error('Something really bad happened');
} catch(e) {
    console.log(e.name + ": " + e.message);
}
```

//Error: Something really bad happened

代码片段 `try-catch.txt`

下面的列表列出了各种不同的 `error` 类型。

- Error

- EvalError
- RangeError
- ReferenceError
- SyntaxError
- TypeError
- URIError

2.7 变量

可以通过赋值操作(隐式声明)或者使用 `var` 关键字(显式声明)来声明变量。如果使用 `var` 关键字来声明变量, 则该变量是持久的, 且不能删除该变量。对于一个变量, 可以多次进行声明, 不必担心产生任何错误。未初始化的变量包含特定的值 `undefined`。



```
i = 0; //隐式声明
var i = 0; //显式声明
var x; //undefined
```

代码片段 `variable-declaration.txt`

作用域(Scope)指的是变量或对象的可见范围。全局变量(global)在应用程序中任何地方都可见, 而局部变量(local)仅在声明该变量的函数内可见。

隐式声明的变量总是具有全局作用域, 即使该变量是在某个函数体中声明的。为了避免可能产生的问题, 建议开发人员总是使用 `var` 关键字来声明变量。JSLint 之父 Douglas Crockford 建议在指定作用域的顶部声明所有需要的变量, 以避免重复定义的问题。如果在函数内和函数外声明了相同的变量, 那么在函数体内局部变量将取代(或者说隐藏)外部声明的变量。



```
function where(){
    var v1 = "local scope";
    v2 = "global scope";
}

where();

console.log( v2 );
console.log( v1 );

//结果
global scope
ReferenceError: v1 is not defined
```

代码片段 `global-scope.txt`

变量名必须以一个字母、下划线或美元符号开头，后跟 0 个或多个字母数字字符、下划线或美元符号。不能使用 JavaScript 语言的关键字作为变量名。

2.8 理解对象

对象是属性的集合，每一个属性具有一个名称和一个值。属性可以包含除 `undefined` 之外的任何类型，在创建了对象之后，可以对属性进行赋值。数组、函数和正则表达式都是对象。数值(`number`)、字符串(`string`)和布尔值(`boolean`)也都是对象，但它们是不可变的对象。可以采用两种方法来实例化对象。第一种方法是使用 `new` 关键字：



```
var myObject = new Object();
```

代码片段 `new-object.txt`

`new` 关键字将调用构造函数，或者更通用的说法是构造器。构造器将初始化新创建的对象。下面的代码块演示了一个名为 `Zombie` 的对象的构造器，构造器初始化该对象的 `name` 属性，然后使用 `new` 关键字实例化一个 `Zombie` 对象。`this` 关键字用于引用当前对象，不能对它进行赋值，但可以将 `this` 关键字的值赋给另外一个变量。



```
//构造函数
function Zombie( name ){
    this.name = name;
}
var smallZombie = new Zombie( "Booger" );
```

代码片段 `this.txt`

另外一种实例化新对象的方法更加方便：使用对象字面量，这种方式创建的对象更像其他语言中的散列(`hash`)或关联数组。



```
var myObject = {};

var objectWithProperties = {
    "property1": "a string value",
    "myObjectAsProperty": myObject
};
```

代码片段 `object-literal.txt`

在属性列表最后一个值的末尾，请不要使用结尾逗号。不同浏览器对此的解释并不一致。既可以使用方括号来访问对象的属性，也可以使用点操作符来访问。下面的代码块演

示了这两种访问方式:



可从
Wrox.com
下载源代码

```
objectWithProperties['property1']
"a string value"
objectWithProperties.property1
"a string value"
```

代码片段 *property-lookups.txt*

采用方括号方式访问属性时,可以使用 JavaScript 的关键字(keyword)作为属性名(不建议采用这种方式),使用点操作符方式访问属性时则不能使用。使用点操作符方式,代码更加简短。JSLint 鼓励开发人员使用点操作符方式。因为属性也可以是对象,可以在任意层次上嵌套对象。



可从
Wrox.com
下载源代码

```
var species = {
  'mammals' : {
    'biped': {
      'monkey' : 'George' ,
      'human' : 'Tim'
    }
  }
}

console.log(species.mammals.biped.human);
```

代码片段 *nested-objects.txt*

JavaScript 是一种动态语言,因此更新一个属性只需要重新对属性赋值即可。要从对象中移除一个属性,只需要使用 `delete` 操作符。删除一个不存在的属性并不会造成任何危险。要遍历对象的所有属性,可以使用 `for...in` 循环,比如下面的代码块:



可从
Wrox.com
下载源代码

```
var obj = {
  'property1' : 1,
  'property2' : 2
}

var i;

for( i in obj ){
  console.log( i );
}

property1
property2
```

代码片段 *for-in.txt*

JavaScript 使用原型继承(prototypal inheritance)，对象直接从其他对象继承，从而创建新的对象。简而言之，对象继承了另外一个对象的属性。JavaScript 中没有类，这是与 Java 和 C#语言相比一个较大的区别。原型就是其他对象的模型(model)。

这听起来似乎很困难，但实际上原型继承比基于类/对象的继承更加简单。一个内置对象 `Object.prototype` 位于继承层次的顶端，所有其他对象都继承了该对象。在继承树中的每一个对象，都可以通过 `prototype` 属性链接起来。

获取属性是通过 `delegation` 实现的。当试图获取一个不存在的属性时，JavaScript 将从原型对象查找该属性。如果依然没有找到该属性，则 JavaScript 将沿着原型树(prototype tree)向上查找，直到查找到原始的 `Object.prototype`。之后，如果依然没有找到该属性，则 JavaScript 解释器将放弃查找并返回一个 `undefined`。

如果对象的某个属性是一个函数，则该函数称为方法。可以通过将函数添加到原型对象来创建方法，也可以直接将函数添加到对象。方法总是具有一个名为 `this` 的参数，它绑定于函数的作用域范围内，`this` 是对调用该方法的对象的一个引用。下面的代码块演示了一个基于原型的继承示例。在后台，创建了一个 `Monster.prototype` 对象。



可从
Wrox.com
下载源代码

```
function Monster( type ){
    this.type = type;
}

Monster.prototype.getType = function(){
    return this.type;
}

function Zombie( name ){
    this.name = name;
}

Zombie.prototype = new Monster();

Zombie.prototype.eatPeople = function(){
    console.log( 'tastes like chicken' );
}
```

代码片段 prototype.txt

引用(reference)是一个指向对象实例位置的指针。`Object` 是引用类型，由于所有对象都是通过引用传递的，修改绑定于一个原型的属性，将修改基于该原型的所有其他对象的原型，下面的代码块演示了这一问题：



可从
Wrox.com
下载源代码

```
> smallZombie.eatPeople();
Tastes like chicken
> delete Zombie.prototype.eatPeople;
true
```



```
> smallZombie.eatPeople();
TypeError: Object #<a Zombie> has no method 'eatPeople'
```

代码片段 `prototype-demo.txt`

当创建新对象时，请注意如何初始化对象。下面的代码说明了两个变量如何引用同一个对象，尽管你可能以为第一行代码是一种简写形式：



可从
Wrox.com
下载源代码

```
//obj1 和 obj2 引用的是同一个对象
var obj1 = obj2 = {};
```

```
//obj1 和 obj2 引用的是不同的对象
var obj1 = {},
    obj2 = {};
```

代码片段 `object-trap.txt`

对象是自知的(self-aware)，或者说对象知道自己的属性。要检查某个属性是否存在，可以使用 `hasOwnProperty()` 方法，该方法将返回一个布尔值。

很多程序员认为，应该避免使用全局变量。有一些办法可以避免扰乱全局名称空间，一种办法是使用单个全局变量作为顶层对象，包含程序或框架中的所有方法和属性。按照惯例，名称空间的字母全部大写，但值得注意的是常量通常也用大写格式。第 10 章将更详细地介绍如何使用对象来组织代码。



可从
Wrox.com
下载源代码

```
ZOMBIE_GENERATOR = {};
ZOMBIE_GENERATOR.Zombies = {
  smallZombie : 'Booger',
  largeZombie : 'Bruce'
}
```

代码片段 `simple-namespace.txt`

采用上面的代码定义名称空间之后，`Zombies` 就不会与全局名称空间中的任何其他变量冲突。另外一种减少冲突的方法是使用闭包，本章随后将介绍闭包的概念。

2.9 使用函数

Douglas Crockford 在他的著作 *JavaScript: the Good Parts* 中声称，函数是 JavaScript 中最优秀的功能。一旦你真正理解了 JavaScript 语言中的函数，就会同意 Douglas Crockford 的观点。JavaScript 中函数的灵活性和威力令人无法忽视。

在使用 jQuery 时尤其如此，jQuery 是一个充分利用了函数强大功能的 JavaScript 库。

函数是一个代码块，它封装了一系列的 JavaScript 语句。函数可以接收参数，也可以返回值。下面的代码示例定义了一个简单的函数语句，它显式地返回一个算术表达式的值。



可从
Wrox.com
下载源代码

```
function calc( x ){  
    return x*2;  
}  
console.log( calc( 10 ) )  
20
```

代码片段 *simple-function.txt*

如果函数没有返回一个特定的值，则它将返回一个 `undefined` 值。下面的代码示例定义了一个没有返回值的函数。该函数依然执行了函数体中的操作，将变量 `x` 的值乘以 2，但由于没有使用 `return` 语句返回值，因此在控制台中输出函数的返回值时，返回值为 `undefined`。



可从
Wrox.com
下载源代码

```
var x = 2;  
  
function calc( ){  
    x = x*2;  
}  
  
console.log( x )  
4  
  
console.log ( calc() );  
undefined
```

代码片段 *undefined-value.txt*

这正是函数有趣的地方。与 C# 和 Java 不同，JavaScript 中的函数是第一类(first-class)的对象。这意味着可以像处理其他 JavaScript 对象一样处理 JavaScript 函数。可以将函数赋予一个变量，或者保存在另外一个数据结构中(比如一个数组或对象)；可以将函数作为参数传递给其他函数；可以将函数作为另一个函数的返回值；函数还可以采用匿名函数的形式：即根本没有绑定于函数名标识符的函数。下面的代码定义了一个函数表达式，并将其赋值给一个变量。



可从
Wrox.com
下载源代码

```
var calc = function(x) {  
    return x*2;  
}  
  
calc( 5 );  
10
```

代码片段 *function-variable-assignment.txt*

给函数名使用圆括号，将执行该函数并返回函数的返回值，而不是返回对函数的引用。



可从
Wrox.com
下载源代码

```
var calc = function(x){
    return x*2;
}

var calcValue = calc( 5 );

console.log( calcValue );
10
```

代码片段 *function-variable-result.txt*

对于 jQuery 开发人员，第一类函数的另外两个特性是非常重要的。第一个特性就是将函数作为参数传递给其他函数。第二个特性就是匿名函数。结合这两个特性，就可以编写出流畅的 jQuery 代码。如果不能在运行时将函数表达式传递给另外一个函数，那么 jQuery 就不称其为 jQuery。当在第 5 章中学习事件处理时，这一点尤为明显。除了处理事件之外，jQuery 中充满了各种可以接收函数作为参数的方法。

下面的代码示例演示了 JavaScript 中函数的重要特性。reporter 函数接收一个函数作为参数，并输出执行该参数函数的返回值。另外两个例子则演示了匿名函数。第一个例子是一个根本不包含任何语句的匿名函数，但根据前面的介绍，该函数将返回一个 undefined。第二个函数是一个返回字符串 a simple string 的匿名函数。



可从
Wrox.com
下载源代码

```
function reporter( fn ){
    console.log( "the return value of your function is "+ fn() );
}

reporter( function(){} );
the return value of your function is undefined

reporter( function(){ return "a simple string" } );
the return value of your function is a simple string

function calc() {
    return 4 * 2
}

reporter( calc );
the return value of your function is 8
```

代码片段 *function-arguments.txt*

关于匿名函数，一个特别重要的变体就是立即调用的函数表达式(immediately invoked function expression, IIFE)，或者称为自执行匿名函数(Self-executing anonymous function)。请参考 Ben Alman 在博客上发表的文章 <http://benalman.com/news/2010/11/immediately-invoked->

function-expression/, 该文章深入地讨论了这种非常常见的模式的名称问题。无论称其为“立即调用的函数表达式”还是“自执行匿名函数”, 这一模式的本质是将一个函数表达式包装在一对圆括号中, 然后立即调用该函数。这一技巧非常简单, 将函数表达式包装在一对圆括号中, 将迫使 JavaScript 引擎将 function(){} 块识别为一个函数表达式, 而不是一个函数语句的开始。下面的代码示例描述了这一模式, 在这个简单的例子中, 函数表达式接受两个值并简单地将二者相加。



可从
Wrox.com
下载源代码

```
(function( x,y ){
    console.log( x+y );
} )( 5,6 );
11
```

代码片段 iife.txt

由于这样的函数表达式将被立即调用, 因此该模式用于确保代码块的执行上下文按照预期的效果执行。这是这种模式最重要的用途之一。通过将参数传入函数, 在执行时就可以捕获函数所创建闭包中的变量。闭包就是这样一个函数: 它处在一个函数体中, 并引用了函数体当前执行上下文中的变量。闭包是一个极为强大的功能, 在学习第 12 章关于一些高级 jQuery 插件模式时, 我们将看到闭包的一些真实应用。就目前而言, 下面的代码示例描述了闭包的基本应用。在下面的例子中还引入了 JavaScript 函数另外一个有趣的特性, 即在函数中可以将另外一个函数作为返回值返回。

在下面的例子中, 将 IIFE 赋值给一个变量 message。message 返回另外一个函数, 该函数只是简单地输出变量 x 的值。有趣的事情是, 当我们把变量 x 的初始值作为参数传入函数时, 可以在函数执行时所创建的闭包中捕获变量 x 的值。无论在外部作用域中 x 的值发生了什么变化, 闭包将记住函数执行时变量 x 的值。



可从
Wrox.com
下载源代码

```
var x = 42;
console.log( x );
var message = (function( x ){
    return function() {
        console.log( "x is " + x );
    }
} )( x );
message();
x = 12;
console.log( x );
message();
```

代码片段 captured-variables.txt

即使只介绍了这个简单的例子, 也应该可以看到 JavaScript 函数的强大功能。在本章后面的小节中, 以及本书后面的章节中, 将进一步介绍 JavaScript 函数的强大功能。

2.10 理解执行上下文

执行上下文是一种对象，代码在执行上下文环境中执行。可以通过 `this` 关键字访问执行上下文，`this` 关键字是对当前执行上下文对象的引用。在目前的 JavaScript 语言版本中，如果代码不存在于一个用户自定义对象或函数中，那么它将属于全局上下文。

请注意，在将来的语言版本中，包括目前在少量浏览器中支持的 JavaScript 版本 5 中的“strict mode”的特殊的语言子集中，没有定义具体上下文的函数，将把它们 `this` 关键字的值绑定于 `undefined` 值。

`eval` 函数和 `setTimeout` 函数也具有自己独立的执行上下文。



```
> console.log(this);  
[object DOMWindow]
```

代码片段 `this-value.txt`

当开始执行一个函数或方法时，JavaScript 创建一个新的执行上下文并进入该函数的执行上下文。当函数执行完毕并返回时，控制权将交还给原来的执行上下文。在创建闭包时当前上下文将继续保留下来，否则当前上下文将被作为垃圾收集。

2.11 作用域和闭包

当讨论作用域时，考虑定义变量的位置和变量的生存期是非常重要的。作用域指的是在什么地方可以访问该变量。在 JavaScript 中，作用域维持在函数级别，而并非块级别。因此，参数以及使用 `var` 关键字定义的变量，仅在当前函数中可见。

除了不能访问 `this` 关键字和参数之外，嵌套函数可以访问外部函数中定义的变量。这一机制是通过闭包来实现的，它指的是：即使在外函数结束执行之后，内部嵌套的函数继续保持它对外部函数的引用。闭包还有助于减少名称空间的冲突。

每次调用一个包裹(enclosed)的函数时，虽然函数的代码并没有改变，但是 JavaScript 将为每一次调用创建一个新的作用域。下面的代码说明了这一行为。



```
function getFunction(value){  
    return function(value){  
        return value;  
    }  
}  
  
var a = getFunction(),  
    b = getFunction(),  
    c = getFunction();  
  
console.log(a(0));  
0
```

```

console.log(b(1));
1

console.log(c(2));
2

console.log(a === b);
false

```

代码片段 `more-closures.txt`

当定义一个独立函数(即不绑定于任何对象)时, `this` 关键字绑定于全局名称空间。作为一个最直接的结果, 当在一个方法内创建一个内部函数时, 内部函数的 `this` 关键字将绑定于全局名称空间, 而不是绑定于该方法。为了解决这一问题, 可以将包裹方法的 `this` 关键字简单地赋值给一个名为 `that` 的中间变量。



```
obj = {};
```

```

obj.method = function(){
    var that = this;
    this.counter = 0;

    var count = function(){
        that.counter += 1;
        console.log(that.counter);
    }

    count();
    count();
    console.log(this.counter);
}

```

```

obj.method();
1
2
2

```

代码片段 `this-that.txt`

2.12 理解访问级别

在 JavaScript 中并没有官方的访问级别语法, JavaScript 没有类似于 Java 语言中的 `private` 或 `protected` 这样的访问级别关键字。默认情况下, 对象中所有成员都是公有和可

访问的。但在 JavaScript 中可以实现与私有或专有属性类似的访问级别效果。要实现私有方法或属性，请使用闭包。



可从
Wrox.com
下载源代码

```
function TimeMachine(){
    //私有成员
    var destination = 'October 5, 1955';

    //公有成员
    this.getDestination= function(){
        return destination;
    };
}

var delorean = new TimeMachine();
console.log(delorean.getDestination());
//October 5, 1955
> console.log(delorean.destination);
//undefined
```

代码片段 *access-levels.txt*

`getDestination` 方法是一个专有方法，它可以访问 `TimeMachine` 中的私有成员。另外，变量 `destination` 是“私有”的，它的值只能通过专有方法 `getDestination` 进行访问。

2.13 使用模块

与私有和专有访问级别类似，JavaScript 没有内置的包语法。模块模式是一种简单而流行的方式，用于创建自包含的、模块化的代码。要创建一个模块，只需要声明一个名称空间、将有关函数绑定于该名称空间，并定义私有成员和专有成员即可，下面将使用模块重写 `TimeMachine` 对象。



可从
Wrox.com
下载源代码

```
//创建名称空间对象
TIMEMACHINE = {};

TIMEMACHINE.createDelorean = (function(){
    //私有成员
    var destination = '';
    var model = '';
    var fuel = '';

    //公有访问方法
    return {
        //设置器
        setDestination: function(dest){
            this.destination = dest;
        }
    };
})();
```



```
    },

    setModel: function(model){
        this.model = model;
    },

    setFuel: function(fuel){
        this.fuel = fuel;
    },

    //访问器
    getDestination: function(){
        return this.destination;
    },

    getModel: function(){
        return this.model;
    },

    getFuel: function(){
        return this.fuel;
    },
    //其他成员
    toString : function(){
        console.log( this.getModel() + ' - Fuel Type: ' +
            this.getFuel() + ' -Headed: ' + this.getDestination());
    }

    //在这里执行初始化过程
};

());

var myTimeMachine = TIMEMACHINE.createDelorean;
myTimeMachine.setModel('1985 Delorean');
myTimeMachine.setFuel('plutonium');
myTimeMachine.setDestination('October 5, 1955');
myTimeMachine.toString();

1985 Delorean - Fuel Type: plutonium - Headed: October 5, 1955
```

代码片段 module.txt

该模块具有一个工厂函数(factory function),它返回一个带有 public/private API 的对象。在第 10 章,还将更详细地介绍模块及其在 jQuery 中的应用。

2.14 使用 JavaScript 数组

数组是一种特殊类型的对象，它作为一个有序的值的集合，这些值称为数组元素。在数组内，每一个元素都具有一个索引，或称为一个数字位置。在一个数组中可以存储相同类型或不同类型的元素。不要求数组元素全都具有相同的数据类型。与对象和函数类似，数组也可以任意嵌套。

与普通对象一样，可以采用两种方式来创建数组。第一种办法是使用 `Array()` 构造函数：



可从
Wrox.com
下载源代码

```
var array1 = new Array(); //空数组
array1[0] = 1;           //在索引为 0 的位置添加一个数组元素
array1[1] = 'a string';  //在索引为 1 的位置添加一个字符串
```

代码片段 `array-constructor.txt`

更常用的是第二种办法，即使用一个数组字面量来创建数组，它由一对方括号括起来，其中包含了 0 个或多个以逗号分隔的值。



可从
Wrox.com
下载源代码

```
var niceArray = [1,2,3];
```

代码片段 `array-literal.txt`

在数组字面量中混合使用对象字面量，可以提供非常强大的结构，它是 JavaScript 对象表示法(JavaScript Object Notation, JSON)的基础。JSON 是一种流行的数据交换格式，它可用于多种语言而不仅仅是 JavaScript 语言。



可从
Wrox.com
下载源代码

```
var myData = {
  'root' : {
    'numbers' : [1, 2, 3],
    'letters' : ['a', 'b', 'c'],
    'mirepoix' : ['tomatoes', 'carrots', 'potatoes']
  }
}
```

代码片段 `json.txt`

数组具有一个 `length` 属性，它的值总是等于数组中元素的个数减去 1。在数组中添加新元素将改变数组的 `length` 属性。要从数组中移除元素，请使用 `delete` 操作符。



可从
Wrox.com
下载源代码

```
>var list = [1, '2', 'blah', {}];
>delete list[0]
>console.log(list);
,2,blah,[object Object]
```



```
>console.log(list.length);  
4
```

代码片段 delete-operator.txt

删除一个数组元素并不会改变数组的长度，只是在数组中留下一个空元素。

2.15 扩展类型

JavaScript 支持将方法和其他属性绑定到内置类型，比如字符串、数值和数组类型。与其他任何对象类似，String 对象也具有 prototype 属性。开发人员可以为 String 对象扩充一些便利的方法。例如，String 没有将字符串 false 或 true 转换为布尔值的方法。开发人员可以使用下面的代码为 String 对象添加该功能：



可从
Wrox.com
下载源代码

```
String.prototype.boolean = function() {  
    return "true" == this;  
};  
  
var t = 'true'.boolean();  
var f = 'false'.boolean();  
  
console.log(t);  
console.log(f);  
  
true  
false
```

代码片段 string-boolean.txt

显然，在每次想为指定类型添加方法时反复输入 prototype 显得有点累赘。Douglas Crockford 为此提供了一段简洁的代码，它使用一个名为 method 的方法扩充了 Function.prototype。下面就是该方法的代码。



可从
Wrox.com
下载源代码

```
Function.prototype.method = function(name, func){  
    this.prototype[name] = func;  
    return this;  
};
```

代码片段 method-method.txt

接下来就可以重写 String 类的 boolean 方法：



可从
Wrox.com
下载源代码

```
String.method('boolean', function(){
    return "true" == this;
});
> "true".boolean();
True
```

代码片段 newBoolean.txt

对于编写工具方法库并将其包含在项目中，这一技术非常有用。

2.16 JavaScript 最佳实践

下面列出了在进行 JavaScript 开发时，一些应该注意或应该避免的事项：

- 使用 `parseInt()` 函数将字符串转换为整数，但请确保总是指定基数。更好的办法是使用一元操作符 `(+)` 将字符串转换为数值。



可从
Wrox.com
下载源代码

```
Good: parseInt("010", 10); //转换为十进制而不是八进制
Better: + "010"           //简单又高效
```

代码片段 cool-conversions.txt

- 使用等同(`===`)操作符来比较两个值。避免意料之外的类型转换。
- 在定义对象字面量时，在最后一个值的结尾不要使用逗号。例如：



可从
Wrox.com
下载源代码

```
var o = {
    "p1" : 1,
    "p2" : 2, //非常糟糕!
}
```

代码片段 trailing-comma.txt

- `eval()` 函数可以接受一个字符串，并将其视为 JavaScript 代码执行。应该限制 `eval()` 函数的使用，它很容易在代码中引入各种各样严重的安全问题。
- 使用分号作为语句的结尾，当精简代码时这特别有用。
- 应该避免使用全局变量。应该总是使用 `var` 关键字来声明变量。将代码包裹在匿名函数中，以避免命名冲突，请使用模块来组织代码。
- 对函数名使用首字母大写表示该函数将作为 `new` 操作符的构造函数，但请不要对其任何其他函数的函数名使用首字母大写。
- 不要使用 `with` 语句。在本书中并未介绍 `with` 语句，因为不应该使用它。
- 在循环中创建函数应该谨慎小心。这种方式非常低效。

2.17 综合示例

根据本章介绍的 JavaScript 知识，我们可以来看一个 jQuery 的示例，不必阅读 jQuery 的文档，就可以从中获得对 jQuery 的初步印象。例如：



可从
Wrox.com
下载源代码

```
$('#submit_button_id').click(function() {  
    $.ajax({  
        'url': '/place2post.php',  
        'type': 'post',  
        'success': function() {  
            //把代码放在这里  
        }  
    });  
});
```

代码片段 `put-it-all-together.txt`

在上面的代码中可以看到很多本章介绍的内容：使用一个对象字面量来传递一个参数列表、匿名函数、美元符号(\$)作为有效的标识符。你也许已经知道，\$是 jQuery 核心对象的别名。不必查看其中任何一行代码就可以认定，在 jQuery 内部使用了闭包来隐藏数据。

2.18 小结

本章介绍了 JavaScript 作为动态语言的特性，还介绍了 JavaScript 一些令人印象深刻的特性。本章简要地介绍了 JavaScript 的数据类型、表达式和面向对象的行为。最为重要的，还介绍了 JavaScript 的函数特性。对于这些主题，每一个都值得深入学习，经过本章的学习，已经为使用 jQuery 奠定了一个坚实的基础。在学习并真正理解了本章的内容之后，你甚至可以打开 jQuery 的源代码，你将看到这些源代码已经不再像之前看到的那样陌生。

2.19 注意

本章中所有代码都使用了 jconsole 进行测试。某些附加的有效命令，比如 print() 和 clear() 也许并未在浏览器中实现。

第 3 章

jQuery 核心技术

本章内容

- jQuery 工具函数
- jQuery 框架的非侵入式(Unobtrusive)JavaScript 结构
- 理解 DOM 和事件

在学习了 JavaScript 的基础知识之后,就可以开始使用 jQuery 了。jQuery 是一个占用存储空间很小的库,它不仅是对 JavaScript 原有混杂功能的一个条理清晰的梳理。jQuery 使选择元素更加简单、将文档的行为从文档结构中分离出来、还可以在运行时操作 DOM。

本章将介绍 jQuery 核心库的基础知识,介绍 jQuery 包装对象并简单介绍一下 jQuery 底层的源代码结构。在阅读 jQuery 的源代码时,使用某种具有折叠代码块功能的 IDE 是非常有用的。本章还回顾了 DOM 元素的概念,以及 jQuery 如何处理事件。

3.1 jQuery 脚本的结构

很多主流程设计语言(这里不提它们的名称)都包含了一个庞大的“标准库”,但它实际上并不是什么标准。在这些程序设计语言中,除了使用一个具有通用工具集的标准库之外,还添加了一些附加的模块。其结果就是需要下载的语言库较大、浪费资源以及很多几乎不会使用到的功能。有鉴于此, jQuery 的创建者经过深思熟虑,剔除了各种类型的“附加”模块,使 jQuery 核心库保持精简和一个较小的尺寸。另外,通过插件系统可以扩展 jQuery,第 12 章将介绍有关插件的内容。

jQuery 的核心函数,通常指的是一个工厂对象,即 jQuery()函数,或者更为常用的别名 \$() 函数。根据第 2 章中介绍的 JavaScript 语言基础,这是一个完全合法的变量名。在这

里, 该变量的类型是 `function`。该函数可以接收一个选择器字符串(selector)、一个原生的 HTML 字符串、一个 DOM 元素、一个已有的 jQuery 对象或者根本无需参数。`$()` 函数的返回值是一个 jQuery 对象, 它具有很多方法, 比如 `$.ajax()` 或 `$.each()`。后面还将介绍更多的方法。

通常情况下, 使用了 jQuery 的脚本将具有下面代码块所定义的结构。请注意, 在使用任何与 jQuery 有关的代码之前, 应该先包含 jQuery 库的 JavaScript 文件。

```
<!DOCTYPE html>
<html>
  <head>
    <script src="jquery.js"></script>
    <script >
      ...将 jquery 代码放在这里
    </script>
  </head>
  <body>
    <!--标记
      不要在这里嵌入 JavaScript
    -->
  </body>
</html>
```

通常情况下, 将 `doctype` 声明放在 HTML 文件的开头, 位于 HTML 标记之前, 以指示该文档的文档类型定义(DTD)。本书将使用 HTML5 的 `doctype` 声明, 它非常简单:

```
<!DOCTYPE html>
```

在绝大多数情况下, 使用 jQuery 的基本模式是: 选择一个 DOM 元素或者选择一个 DOM 元素的集合、对选中的元素(或元素集)执行某种操作或控制。选择和操作, 反复执行这样的步骤, 就是 jQuery 的基本工作过程。调用 jQuery 的基本方式是:

```
$(selector);
```

在上面的代码中, `selector`(选择器)是一个字符串组成的表达式, 用于匹配 DOM 元素。jQuery 选择器的格式与 CSS 选择器类似, 因此开发人员可以将使用 CSS 选择器的经验应用于 jQuery 选择器。例如, `#` 用于匹配元素的 `id` 属性, `.` 用于匹配 CSS 类。要选择一个 ID 为 `feed` 的 `div` 元素, 可以使用下面的代码:

```
$("div#feed");
```

第 4 章将更详细地介绍 jQuery 中的选择器。

在使用 jQuery 对象选取元素或者创建新元素时, 它返回的值是一个包装对象。之所以称之为包装对象, 是因为它在 DOM 元素的集合上“包装”了 jQuery 的功能。一个包装对象类似于一个数组, 可以使用方括号索引包装对象, 或者使用它的 `.length` 属性检查元素的数量。开发人员常常使用 `.length` 属性来检查选择器是否匹配了一个元素集。使用 Firebug

的控制台可以轻而易举地查看.length 属性。

当调用一个 jQuery 方法——比如调用.addClass()方法时, jQuery 将把.addClass()方法应用于选中的所有元素。不必使用循环来遍历元素的集合。

前面曾经介绍过, jQuery 对象包含了一些方法。表 3-1 列出了其中一些方法。

表 3-1 jQuery 方法

方 法	分 类	描 述
.ready()	事件	声明一个函数, 当 DOM 完全加载后运行该函数
.click()	事件	为匹配的元素集设置 click 事件处理程序
.ajax()	Ajax	jQuery 的 Ajax 工具函数
.addClass()	CSS	为匹配的元素集添加一个 CSS 类
.removeClass()	CSS	从匹配的元素集中移除一个 CSS 类
.attr()	Attributes	获取或设置指定属性的值
.html()	Attributes、 DOM 插入	获取或设置匹配的元素的 HTML 内容
.type()	工具方法	判断一个对象在 JavaScript 内部的类型[class]

jQuery 是生态友好的, 除了 jQuery 和\$之外, 它不会污染全局名称空间。虽然其他 JavaScript 库中也可能会使用\$别名, 但在其他任何库中几乎不会使用 jQuery 作为变量名。

3.1.1 工具函数

根据第 2 章的介绍, JavaScript 中的函数也是一种对象。因此, 函数也可能具有自己的属性和方法。jQuery 对象具有一些非常有用的方法, 简称为工具方法。工具方法可以增强对于数组、对象、函数甚至数据的操作。

1. 对象

要检查对象的类型, 普通的老式 JavaScript 办法是使用 typeof()操作符, 但它有时不合逻辑。在某些情况下, typeof()返回一个不正确的值, 或者返回一个出乎意料的值。比如 typeof(null)——它返回 object 而不是 null。幸运的是, jQuery 提供了一个自定义的.type()方法, 解决了这些问题:

```
$.type(null); //返回 null
$.type([]); //返回 array
```

.isEmptyObject()方法用于检查一个对象是否包含任何属性, 包括继承的属性, 它与对象的类型无关:

```
$.isEmptyObject(""); //返回 true
$.isEmptyObject({}); //返回 true
```



```
var mailman = {};
mailman.letters = 100;
$.isEmptyObject(mailman); //返回 false
```

与之类似的是.isPlainObject()方法。它也用于检查一个对象是否包含任何属性，但它所检查的必须是一个 Object 的实例。因此对于该函数，对空字符串进行检查将返回 false：

```
$.isPlainObject(""); //false
$.isPlainObject({}); //true
$.isPlainObject(new Object); //true
```

对于对象，jQuery 还能执行哪些操作呢？可以使用.extend()方法来合并两个或两个以上的对象。值得注意的是，.extend()方法的功能是将一个或多个对象的属性合并到一个目标对象之中。在下面的例子中，obj1 将获得 obj2 和 obj3 的属性：

```
var obj1 = {"1":"property 1"};
var obj2 = {"2":"property 2"};
var obj3 = {"3":"property 3"};
$.extend(obj1, obj2, obj3);
console.log(obj1["3"]); //显示属性"3"的值
```

关于.extend()方法，有趣的是可以使用它来克隆 JavaScript 对象(这与\$.clone()方法不同)：

```
var clonedObject = $.extend({}, anObject);
```

\$.extend()方法还可以接收一个布尔值作为第一个参数，用于执行对象的深度合并，即递归复制。可以使用该方法来实现对象的深度克隆：

```
var clonedObject = $.extend(true, {}, anObject);
```

2. 函数

jQuery()具有两个用于函数的工具方法：.isFunction() 和.noop()。顾名思义，.isFunction()方法用于检查一个对象是否是一个函数。在使用.isFunction()方法时，请确保去掉函数名的圆括号。在下面的例子中，将.isFunction()函数作为参数传入.isFunction()函数自身。由于它是一个函数，因此返回值必然是 true。

```
$.isFunction({}); //false
$.isFunction($.isFunction); //true
$.isFunction($.isFunction()); //false，由于圆括号
```

.noop()方法是一个存根函数(stub function)，它并不执行任何操作。但在某些情况下它是非常有用的，比如当我们想把一个空函数作为参数传递时，以便安装一个在默认情况下不执行任何操作的新事件。

3. 数组操作

与.isObject()方法和.isFunction()方法类似,对于数组也具有一个.isArray()方法。除了检查一个对象是否是一个数组之外,还可以使用.makeArray()方法将一个类似于数组的对象转换为一个真正的数组。将类似于数组的对象转换为数组,将丢失该对象的所有方法和属性。下面的代码描述了使用.makeArray()方法的一个例子,它将一个克隆版的 jQuery 对象转换为一个标准的数组。

```
//克隆 jQuery 对象
var clonedjq = $.extend(true, {}, $);
//返回"object"
$.type(clonedjq);

//转换为一个 JS 数组
var jqArray = $.makeArray(clonedjq);

//返回"array"
$.type(jqArray);
```

jQuery 还提供了另外一个便捷方法用于合并数组,该方法称为.merge()方法。它从第 2 个数组中获取元素,并将这些元素追加到第一个数组中,并保留两个数组中元素的原有顺序:

```
var spooks = ["wraiths", "vampires", "lichs"];
var nums = [1,2,3];
var mergedArrays = $.merge(spooks, nums);
alert(mergedArrays); //显示结果: wraiths, vampires, lichs, 1, 2, 3
```

接下来介绍的两个方法,对于操作数组中的元素非常有用。.isArray()方法用于检查数组中是否存在某个指定的值,如果找到该值则返回它在数组中的索引。如果该值不在数组中,则返回-1。\$.unique()方法的功能是从 DOM 元素的数组中移除重复的元素。下面的代码块演示了这两个方法的应用:



```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <p></p>
    <p></p>
    <p></p>
    <p class="duplicate"></p>
    <p class="duplicate"></p>
    <p class="duplicate"></p>
    <p></p>
    <script
      src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min
```

```

.js">
</script>
<script>
$(document).ready(function(){
//选择所有的<P>元素
var paragraphs = $( "p" );

//返回 7
console.log( paragraphs.length );

//将其转换为一个数组
paragraphs = $.makeArray( paragraphs );

//获取类名为 duplicate 的元素，并立即将其转换为一个数组
var dupes = $.makeArray$( ".duplicate" ))

//将 dupes 数组连接到 paragraphs 数组
paragraphs = paragraphs.concat( dupes );

//返回 10
console.log( paragraphs.length );

//去除重复元素，即去除 dupes 数组中的元素
paragraphs = $.unique( paragraphs );

//返回 7
console.log( paragraphs.length );

var index = $.inArray("6", paragraphs);
console.log(index); //返回-1
});
</script>
</body>
</html>

```

代码片段 in-array-and-unique.txt

在上面的代码中，首先使用一个 jQuery 选择器选取了所有的<p>元素，然后使用.makeArray()将其转换为一个纯粹的 JavaScript 数组。接下来，使用了一个 jQuery 类选择器选取所有 CSS 类名为 duplicate 的元素，并将它们添加到之前创建的 paragraphs 数组中，这样一来，paragraphs 数组中包含了 3 个重复元素，它的总长度为 10。使用.unique()方法，可以删除 paragraphs 数组中的重复元素，执行.unique()方法后 paragraphs 数组的总长度为 7。最后，检查了 paragraphs 数组中是否包含 6 这个元素，显然它的返回值为-1。

我们可以使用旧式的 for 循环来遍历一个数组，但你可能已经猜到，jQuery 已经提供了一个名为.each()的方法，它可以取代旧式的 for 循环语句。each()方法可以用于遍历对象、类似于数组的对象和数组。\$.each()方法的索引是基于 0 的，它接收两个参数：第一个参数是要遍历的集合，第二个参数是一个回调函数。该回调函数接收如下参数：第一个参数是

当前遍历元素在集合中的索引，第二个参数是该元素的值。下面的代码演示了如何使用 `$.each()` 方法遍历一个简单数组的成员：



```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript">
      $.ready(function(){
        var numArray = [1,2,3,4];
        $.each(numArray, function(index, value){
          value = value * 2;
          console.log("index is:"+index+" new value is:"+value);
        });
      });
    </script>
  </head>
  <body>
  </body>
</html>
```

代码片段 `each.txt`

接下来再介绍一个与数组有关的方法，即 `.map()` 方法。它的语法与 `.each()` 方法类似，它接收两个参数，第一个参数是一个数组对象或者类似于数组的对象，第二个参数是一个回调函数。根据 jQuery 的文档，该回调函数是“对每一个元素项进行处理的函数”，该回调函数接收两个参数，第一个参数是当前元素的值，第二个参数是当前元素在数组中的索引。



```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript">
      $.ready(function(){
        var numArray = [42,1024,486,109821];
        console.log(numArray);
        var newArray = $.map(numArray, function(value, index){
          return Math.sqrt(value).toFixed(4);
        });
        //输出日志 ["6.4807", "32.0000", "22.0454", "331.3925"]
        console.log(newArray);
      });
    </script>
  </head>
  <body>
  </body>
</html>
```

代码片段 `map.txt`

4. 数据结构

队列是一种先进先出(First In, First Out)的数据结构, 队列总是将元素添加到队列的尾部, 移除元素时则总是从队列的头部开始。在 jQuery 中, 可以使用 `$.queue()` 来维护一个函数的列表。默认的函数队列是 `fx`, 它是一个具有标准功能的队列。在 `fx` 的上下文中, 队列的功能是显而易见的。下面的代码示例描述了一个简单的动画队列。ID 为 `animation` 的 `div` 元素先滑入(`slideUp`), 在滑入动画执行完成后, 函数队列中的下一个动画淡入(`fadeIn`)开始运行。

```
$('#animation').slideUp().fadeIn();
```

`$.queue` 队列支持 `push` 和 `pop` 操作。要从队列中移除所有函数, 可以使用 `.clearQueue()` 方法。使用 `.dequeue()` 方法, 则可以从队列中移除一个函数并执行它。下面的代码示例演示了一个使用 `.queue()` 和 `.dequeue()` 方法的简单例子。首先将第二个动画添加到队列中, 然后使用 `.dequeue()` 方法, 从队列栈中弹出一个函数并执行它。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
  <script type='text/javascript'
    src='http://code.jquery.com/jquery-1.7.1.js'></script>
  <style type='text/css'>
    #animation {
      width:200px;
      height:200px;
      background:#ccc;
    }
  </style>

  <script type='text/javascript'>
    $.ready (function() {
      $("#animation").show("slow");
      $("#animation").queue(function () {
        $(this).animate({width:'+=400'},1000);
        $(this).dequeue();
      });
    });
  </script>
</head>
<body>
<div id ="animation" style="display:none"></div>
</body>
</html>
```

代码片段 `queue-and-dequeue.txt`

5. 字符串

对于字符串仅介绍一个工具方法.trim()。JavaScript 已经包含了一种强大的机制，可以通过正则表达式来处理字符串，但 JavaScript 内置的 String 对象并不包含.trim()方法。在下面的代码中，.trim()方法用于移除字符串首尾的空格字符。

```
var phrase1 = "    1. Either this wallpaper goes    ";
var phrase2 = " or I go.                                ";

//Oscar Wilde 的遗言
var quote = $.trim(phrase1) + " " + $.trim(phrase2);
console.log(quote);
//输出 1. Either this wallpaper goes or I go.
```

6. 数据

Ajax 是现代 Web 应用的一个重要组成部分。Ajax 可以实现这样的功能：把客户端的数据传递给服务器，或者在客户端获取服务器的数据，而不必刷新整个页面。Ajax 为现代 Web 应用程序提供了优秀的可用性。在 Ajax 中有两种最主要的数据交换格式，一是 XML，二是 JSON(JavaScript 对象表示法)。jQuery 可以使用.parseXML()和.parseJSON()方法来解析这两种格式的数据，它既可以解析本地数据，也可以解析 URL 指定的数据。在本书介绍 Ajax 的章节将深入学习这两个方法。

可以将任意数据附加到 HTML 元素，可以存储或取回数据。这正是工具方法.data()的功能。在第 6 章中将详细介绍.data()方法。

7. 其他有用的工具方法

下面将要介绍的这些工具方法不适于归入其他类别：.contains()、.isWindow()、.isXMLDoc()、.now()、.support()和.globalEval()方法。这一小节将逐一介绍这些方法。

检查子节点

.contains()方法用于检查一个 DOM 节点是否是另外一个节点的子节点。该方法接收两个参数——第一个参数是容器节点，第二个参数是要检查的目标节点。

```
$.contains($("#head")[0], $("#title")[0]); //返回 true
```

jQuery 包装器返回的是一个类似于数组的对象，因此要引用实际的元素，还需要使用方括号和索引来获取对元素的引用。<title>标记总是<head>标记的子元素，因此该.contains()方法的返回值为 true。

窗口检查

如果我们考虑使用了 iframe 的情形。在某个操作中，我们可能需要区分 iframe 和浏览器的窗口，此时可以使用.isWindow()方法来实现该功能：



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript">
      $.ready(function(){
        var inlineFrame = $("#frame")[0];
        $.isWindow(inlineFrame); //返回 false
        $.isWindow(window); //返回 true
      });
    </script>
  </head>
  <body>
    <iframe id="frame" src="externalPage.html"></iframe>
  </body>
</html>
```

代码片段 isWindow.txt

获取当前时间

顾名思义，快捷方法\$.now()用于获取当前时间。它是(new Date).getTime()方法的一个快捷方式，显然使用\$.now()非常简洁。

检查浏览器特性

开发人员普遍认为，检查浏览器的特性要比检查浏览器更加准确。要检查浏览器所支持的特性，可以使用 jQuery 的.support()方法。该方法用于取代较低 jQuery 版本中的\$.browser。一个很好的例子是：if (\$.support.ajax) {}，它可以检查浏览器是否支持 Ajax 请求(亦称为创建 XMLHttpRequest 对象)。

在全局上下文中求值

这里要介绍的最后一个工具方法是.globalEval()方法。请回忆一下 JavaScript 中的 eval()表达式，可以使用 eval()方法来执行任意的 JavaScript 代码：

```
//在所有 DOM 元素加载完之后，创建一个变量 x 并将它的值初始化为 0
$(function(){
  eval("var x = 0;");
});
```

eval()方法的执行上下文并不是全局的，而是局部的。有时我们希望将用于求值的语句放在全局上下文中，例如当加载一个外部 JavaScript 文件时。这时应该使用.globalEval()方法：

```
//在全局上下文中创建变量
$(function(){
  $.globalEval("var x = 0;");
});
```

表 3-2 汇总了我们所讨论的所有工具方法。

表 3-2 工具方法汇总

方 法	分 类	描 述
<code>\$.type()</code>	工具方法	判断对象在 JavaScript 语言内部的类型[class]
<code>\$.isEmptyObject()</code>	工具方法	检查对象是否为空(即不包含任何属性)
<code>\$.isPlainObject()</code>	工具方法	检查对象是否是一个纯对象(plain object), 即使用 {} 或者 new Object 创建的对象
<code>\$.extend()</code>	工具方法	将两个或两个以上的对象的内容合并到第一个对象中
<code>\$.isFunction()</code>	工具方法	判断传递给该方法的参数是否是一个 JavaScript 函数对象
<code>\$.noop()</code>	工具方法	表示一个空函数
<code>\$.inArray()</code>	工具方法	搜索特定的值是否存在于数组中, 如果找到的话返回它在数组中的索引, 如果找不到则返回-1
<code>\$.isArray()</code>	工具方法	判断传递给该方法的参数是否是一个数组
<code>\$.makeArray()</code>	工具方法	将一个类似于数组的对象转换为一个真正的 JavaScript 数组
<code>\$.merge()</code>	工具方法	将第 2 个数组的内容合并到第一个数组中
<code>\$.map()</code>	工具方法	将当前匹配元素集中的每一个元素传递给一个函数, 产生一个新的 jQuery 对象, 它包含了用函数处理每一个元素的返回值
<code>\$.each()</code>	工具方法	一个通用的迭代函数, 它可以无缝地在对象和数组上执行迭代。数组和类似于数组的对象都具有一个 length 属性(比如函数的 arguments 对象), 可以通过数值索引从 0 到 length -1 进行迭代。其他对象可以通过它的命名属性进行迭代
<code>\$.unique()</code>	工具方法	对 DOM 元素的数组进行排序, 同时删除重复的元素。请注意, 该方法仅对 DOM 元素的数组有效, 对字符串或数值数组无效
<code>\$.queue()</code>	数据、工具方法	在匹配的元素上显示将要执行的函数队列
<code>\$.clearQueue()</code>	自定义、数据、工具方法	从队列中移除所有还未被运行的函数
<code>\$.dequeue()</code>	数据、工具方法	对于匹配的元素, 执行队列中的下一个函数
<code>\$.trim()</code>	工具方法	移除字符串首尾的空格
<code>\$.grep()</code>	工具方法	在数组中查找满足某个筛选函数的元素, 原数组不会受影响
<code>\$.contains()</code>	工具方法	检查一个 DOM 元素是否是另外一个 DOM 元素的子元素

(续表)

方 法	分 类	描 述
\$.data()	数据、 工具方法	存储与特定元素相关的任意数据，返回所设置的值
\$.parseXML()	工具方法	将一个字符串解析为一个 XML 文档
\$.parseJSON()	工具方法	接收一个格式良好的 JSON 字符串，并返回一个对应的 JavaScript 对象
\$.isWindow()	工具方法	判断传递给该方法的参数是否是一个浏览器窗口
\$.isXMLDoc()	工具方法	检查一个 DOM 节点是否属于一个 XML 文档(或者该 DOM 节点是否是一个 XML 文档)
\$.now()	工具方法	返回一个表示当前时间的数值
\$.support()	全局 jQuery 对象的属性、 工具方法	一个属性的集合，表示不同浏览器的特性或存在的 bug
\$.globalEval()	工具方法	在全局上下文中执行某些 JavaScript 代码

3.2 非侵扰式 JavaScript

在 JavaScript 的“黑暗”年代，类似于下面的代码很常见：



```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      function showStuff(){
        var content = document.getElementById("content");
        content.style.display = "block";
      }
      function changeBGColor(elem){
        elem.style.backgroundColor = "green";
      }
    </script>
  </head>
  <body>
    <ul id="content" onLoad="javascript:showStuff();" style="display:none;">
      <li onClick="javascript: changeBGColor(this);">Item 1</li>
      <li onClick="javascript: changeBGColor(this);">Item 2</li>
      <li onClick="javascript: changeBGColor(this);">Item 3</li>
      <li onClick="javascript: changeBGColor(this);">Item 4</li>
    </ul>
  </body>
```



```
</html>
```

代码片段 *dark-days.txt*

看到在标记中嵌入 CSS 风格的代码,你可能会感到厌恶。在基于标准的 Web 开发中,一个基本的信条是:用 CSS 样式定义的表现层信息,应该从由 HTML 定义的页面内容和结构中分离出来。对于上面的例子,要实现样式与内容的分离,应该将样式表放在一个外部 CSS 文件中。改进后的例子如下所示:



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <style type="text/css" src="myCss.css"></style>
    <script type="text/javascript">
      function showStuff(){
        var content = document.getElementById("content");
        content.style.display = "block";
      }
      function changeBGColor(elem){
        elem.style.backgroundColor = "green";
      }
    </script>
  </head>
  <body>
    <ul id="content" onLoad="javascript:showStuff();" class="contentClass">
      <li onClick="javascript: changeBGColor(this);">Item 1</li>
      <li onClick="javascript: changeBGColor(this);">Item 2</li>
      <li onClick="javascript: changeBGColor(this);">Item 3</li>
      <li onClick="javascript: changeBGColor(this);">Item 4</li>
    </ul>
  </body>
</html>
```

代码片段 *separation-of-content-and-style.txt*

将标记内容(HTML)和样式(CSS)分离的原则,也适用于行为(JavaScript)。表示页面行为的代码,也应该从页面内容中分离出来。开发人员应尽量避免使用内联事件处理程序和内联 JavaScript 代码。将行为与内容分离,这种方式称为非侵扰式 JavaScript(unobtrusive JavaScript)。下面的代码示例重写了上面的例子,将样式、内容和行为完全分离。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <style type="text/css" src="myCss.css"></style>
    <script>
      (function(document) {
```

```

        document.onload = function(){
            content.style.display = "block";
        }

        var listItems = document.getElementsByTagName("li");
        for(i = 0; i < listItems.length; i++){
            listItems[i].onclick = function{
                listItems[i].style.backgroundColor = "green";
            }
        }
    })(document);
</script>
</head>
<body>
    <ul id="content" class="contentClass">
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
        <li>Item 4</li>
    </ul>
</body>
</html>

```

代码片段 separation.txt

这样的代码看起来更加优美——HTML 是语义化的，并且 JavaScript 是非侵扰式的。标记就是纯粹的标记，不包含任何行为。但是可以看到，其中的 JavaScript 代码有点繁琐。使用 jQuery 可以使 JavaScript 代码更加简洁：



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
    <head>
        <style type="text/css" src="myCss.css"></style>
        <script src="jquery.js"></script>
        <script>
            $(document).ready(function(){
                $("#content").show();
                $("li").click(function(this){
                    this.css("backgroundColor", "green");
                });
            });
        </script>
    </head>
    <body>
        <ul id="content" class="contentClass">
            <li>Item 1</li>
            <li>Item 2</li>
            <li>Item 3</li>

```

```

        <li>Item 4</li>
    </ul>
</body>
</html>

```

代码片段 unobtrusive.jquery.txt

显而易见, JavaScript 代码从 11 行精简到 7 行, 而且更加简单。在代码中将文档的 DOM 元素传入 jQuery() 函数。jQuery 将包装该 DOM 元素, 然后返回一个包装过的 jQuery 对象, 并应用一个 .ready() 方法。

.ready() 方法用于注册一个处理程序, 一旦文档的所有 DOM 元素加载完毕后就执行该处理程序中的代码, 不必等待所有页面元素(比如图片或 Flash 文件)都下载完毕。.ready() 方法接收一个函数作为其参数。例如, 在下面的代码示例中定义了一个 functionToExecute() 函数, 并将该函数传递给 .ready() 方法, 尽管这种方式从技术上来说可行, 但将一个仅执行一次的函数添加到名称空间之中, 具有一定的副作用, 我们希望消除这种副作用。

```

function functionToExecute() {
    // 执行所需操作
}
$(document).ready(functionToExecute);

```

使用匿名函数来代替 functionToExecute() 函数是最好的办法。

```

$(document).ready(function() {
    // 执行所需操作
});

```

在上面的例子中, 当文档的 DOM 加载完毕后, 使无序列表的内容变为可见, 然后为文档中所有 元素添加一个 onClick 事件处理程序, 在该事件处理程序中使用 .css() 方法, 改变当前 元素的背景色。

对于本例中的 .ready() 方法(或称为事件), 也可以采用下面的形式:

```

$(function() {
    // 将 ready 事件的处理代码放在这里
});

```

这在功能上是等价的, 而且更加简短。在本书后面的内容中, 将采用这种简洁的表示形式。

最后, 将 JavaScript 代码移到一个名为 unobtrusive.js 的独立文件中:



```

<!DOCTYPE html>
<html>
  <head>
    <style type="text/css" src="myCss.css"></style>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript" src="unobtrusive.js"></script>

```



```

</head>
<body>
  <ul id="content" class="contentClass">
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
    <li>Item 4</li>
  </ul>
</body>
</html>

```

代码片段 *full-separation.txt*

采用这种方式，美工人员可以完成他的工作，而不会影响到页面的标记代码，程序设计人员也可以完成自己的工作，不会与美工人员因为修改问题产生冲突。

现在是介绍链式操作的好时机，链式操作是 jQuery 中最常使用的特性之一。请考虑下面的代码：



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
  <head>
    <style type="text/css">
      .class1 {
        color: "white";
        background-color: "black";
        width:200px;
        height:100px;
      }
      .class2 {
        color: "yellow";
        background-color: "red";
        width:100px;
        height:200px;
      }
    </style>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript">
      $(function(){
        $("#myDiv").addClass("class1");
        $("#p#blah").removeClass("class1");
        $("#p#blah").addClass("class1");
      });
    </script>
  </head>
  <body>
    <div id="myDiv">
      <p id="lorem">Lorem Ipsum</p>
      <p id="blah">blah blah blah</p>
    </div>
  </body>
</html>

```

```

    </div>
  </body>
</html>

```

代码片段 no-chaining.txt

在上面的代码中，调用了三个 jQuery 函数，其中的每一个函数都从文档的 DOM 中选取一个元素并执行某种操作，比如将一个 CSS 类添加到元素中，或者从元素中移除一个 CSS 类。实际上，这段代码并没有任何错误，但还有更加高效的办法。一个对带有选择器参数的函数调用，将返回一个 jQuery 对象的实例，jQuery 对象都包装了一系列选择 DOM 元素的方法。

一些 jQuery 方法也返回包装对象，例如：

```
$('#body').find('div');
```

由于 `.find()` 方法返回了另外一个包装对象，因此可以连续地执行另外一个方法调用，比如下面的代码：

```
$('#body').find('div').addClass('cssClass');
```

这称为调用链。在 jQuery 开发中，较长的 jQuery 方法链是很常见的。采用链式操作，除了可读性更好之外，它还具有更好的性能。因为链式操作不必再实例化一个新的 jQuery 对象、搜索 DOM 然后调用一个方法。在下面的代码中，使用链式操作代替了单独操作，重写了上面的例子：



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
  <head>
    <style type="text/css">
      .class1 {
        color: "white";
        background-color: "black";
        width:200px;
        height:100px;
      }
      .class2 {
        color: "yellow";
        background-color: "red";
        width:100px;
        height:200px;
      }
    </style>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript">
      $(function(){
        $("#myDiv")
          .addClass("class1")

```

```

        .find("p#blah")
        .removeClass("class1")
        .addClass("class1");
    });
</script>
</head>
<body>
    <div id="myDiv">
        <p id="lorem">Lorem Ipsum</p>
        <p id="blah">blah blah blah</p>
    </div>
</body>
</html>

```

代码片段 chaining.txt

根据 Cody Lindley 的术语, 对于任何返回另外一个包装对象的 jQuery 方法, 由于它可以继续构造一个调用链, 因此将其定义为一个“构造性”方法; 任何不返回包装对象的方法(比如.text()或.html()方法), 则将其定义为“解构性”方法。

使用 jQuery 精简 JavaScript 代码

就自身的功能而言, JavaScript 是一门非常强大的语言, 但有时 JavaScript 代码显得很繁琐。诸如 jQuery 这样的 JavaScript 框架提供了大量快捷方法, 以减少用于完成常见 Web 开发任务所需的代码数量。比如下面的例子:

JAVASCRIPT 方式:

```

window.onload = function() {
    var elements = document.getElementsByTagName("div");
    for(var i = 0; i < elements.length; i++){
        elements[i].style.color = green;
    }
};

```

JQUERY 方式:

```

$(function(){
    $("div").each(function(index, value){
        $( this ).css( "color" , "green" );
    });
});

```

对于以上的代码, 虽然二者在代码行的数量上差别并不明显, 但是采用 jQuery 方式只需要编写更少的代码, 而且 jQuery 方式更加简明扼要。很快, 开发人员就会习惯使用 \$(function){...};来取代 window.onload()。

当然, 使用诸如 jQuery 这样的 JavaScript 库还有其他更重要的优势。简洁的代码固然很好, 但能以跨浏览器方式添加许多强大的功能则更为重要。在上面这个具体的例子中,

jQuery 以一种跨浏览器方式提供了为 `onLoad` 事件设置多个回调函数的能力。如果没有 jQuery 的跨浏览器解决方案,那么只能为 `onLoad` 事件设置唯一一个回调函数,如果不小心还可能无意中覆盖之前设置的回调函数。

顺便介绍一下,这一问题最初是由一个技术社区解决的,该社区主要以富有影响力的 JavaScript 开发者 Dean Edwards 发表的一系列博客为主,这些博客包括:<http://dean.edwards.name/weblog/2005/09/busted> 和 <http://dean.edwards.name/weblog/2006/06/again/>。对于完美解决方案最重要的贡献者,就是 jQuery 的创建者 John Resig 本人。

3.3 jQuery 框架的结构

jQuery 是一个精心编写的模块化框架,使用者很容易将 jQuery 视为一个“黑盒(black box)”。但是,学习 jQuery 底层的源代码、研究 jQuery 中的各种功能是如何实现的,这是非常有价值的。截至 jQuery 1.3,选择器引擎已经分离出来,基于一个 John Resig 发起的独立项目 `sizzle`。比起之前版本中选择元素的方法,它在性能上进行了一些增强。很多信息请参考 <http://sizzlejs.com>。

如果回顾一下第 2 章,一个自调用匿名函数具有如下形式:

```
(function(arguments){
    //执行某些操作
})();
```

jQuery 也是一个自执行的匿名函数,它接收两个参数——第一个参数是 `window` 对象,第二个参数是 `undefined`:

```
(function( window, undefined)
{...} //jQuery 核心定义
)( window);
```

将 `window` 对象作为参数传入 `jQuery()` 函数,有助于提高 jQuery 的性能。这样一来,在 jQuery 函数的定义中,对于任何对全局 `window` 对象的调用,就不必在全局上下文中查找 `window` 对象的定义,只需要在局部上下文中查找即可,减少了一个查找步骤。虽然对性能的提升很小,但它对改进性能确实有用。

将第二个参数 `undefined` 传入 `jQuery()` 函数,看起来似乎令人遗憾,但这是一种很好的方式。内置类型 `undefined` 是可以重新赋值的,因此可以确信开发团队中的某人可能会执行类似于 `undefined = true`; 这样的代码,它会造成比较操作无法得到的预期结果。(对于团队成员,Paul Irish 使用了一个特殊的术语来描述这一状况。请参考他的视频 `10 Things I Learned from the jQuery Source`,访问该视频的地址是 <http://paulirish.com/2010/10-things-i-learned-from-the-jquery-source/>。

继续查看源代码,可以看到各种工具方法是如何添加到 jQuery 函数中的:

```

...
jQuery.extend({
    noConflict: function( deep ){
        ...
        isFunction: function( obj ){
            ...
            isArray: Array.isArray || function( obj ){
                ...
            }
        }
    }
});

```

从上面这段 jQuery 源代码可以看到，jQuery 调用了函数.extend()，它接收一个对象字面量作为参数，该对象字面量用键/值对来表示方法的名称和方法的定义。仔细阅读这些方法的定义，可以确切地看到比较操作是如何执行的、以及每一个方法的功能。

在查看 jQuery 源代码时，最有意思的是学习\$.ready()方法是如何工作的。在 DOM 中并没有.onReady()事件。学习\$.ready()方法的源代码非常有价值：

```

//在 Internet Explorer 中检查 DOM 是否就绪 (DOM ready)
function doScrollCheck() {
    if ( jQuery.isReady ) {
        return;
    }
    try {
        //如果是 IE 浏览器，则使用 Diego Perini 的技巧进行处理
        //请参考 http://javascript.nwbox.com/IEContentLoaded/
        document.documentElement.doScroll("left");
    } catch(e) {
        setTimeout( doScrollCheck, 1 );
        return;
    }

    //执行任何等待执行的函数
    jQuery.ready();
}

//将 jQuery 对象暴露为全局对象
return jQuery;
...

```

一旦所有 DOM 元素加载完毕就会执行.ready()方法，不必等待页面中所有部分都下载完成，比如图片或 Flash 文件。在 DOM 元素加载完毕后，就可以操作 DOM 了。如果在 DOM 还未加载完毕时，试图运行.doScroll()方法将抛出一个异常。在上面的 jQuery 源代码中，把对 doScroll()方法的调用放在一个 try/catch 语句之中。当异常发生时(即 DOM 元素正在加载时)，doScrollCheck()方法将递归调用自身，直到 DOM 完全加载。此时 doScroll 方法停止抛出异常，jQuery 将执行绑定到.ready()的代码。

这一小节很简短，无法真正体现 jQuery 源代码中各种强大的编程技术。强烈建议学习 jQuery 的源代码，源代码中的很多技术可以应用于我们自己的项目中。

3.4 理解 DOM 和事件

HTML 和 XML 的标记(或元素)都是层次型的树结构。浏览器窗口包含了一个文档,文档包含了一个<HTML>元素,它又依次包含了子元素<header>和<body>,依此类推。W3C 规范对 DOM 的定义是最完美的:

“文档对象模型(Document Object Model)是一个中立足于平台和语言的接口,它允许程序和脚本动态地访问和更新文档的内容、结构和样式。可以对文档进行进一步的处理,并且处理结果可以合并回呈现该文档的页面。”

在 jQuery 中,绝大多数与 DOM 有关的跨浏览器差异的问题,都被 jQuery 默默处理了。开发人员不必过多地担心 Gecko 友好浏览器、IE、Safari 和 Opera 浏览器在处理 DOM 问题上的差异。

3.5 与其他 JavaScript 库一起使用 jQuery

有时开发人员希望在程序中包含其他 JavaScript 框架,或在代码中使用其他 JavaScript 库。如果这些框架使用了美元符号(\$)作为变量名,就会导致与 jQuery 的冲突。但是 jQuery 可以与其他 JavaScript 库很好地协作, jQuery 具有避免这种冲突的方法。在加载了其他可能导致冲突的 JavaScript 库之后,为了避免 jQuery 与之冲突,只需要使用\$.noConflict();方法。该方法的功能是:将美元符号恢复为加载 jQuery 之前的值。在调用了.noConflict()方法之后,使用 jQuery 来代替\$,以引用 jQuery 核心对象。下面的代码演示了一个使用.noConflict()方法的例子:

```
<html>
  <head>
    <script src="conflictingFramework.js"></script>
    <script src="jquery.js"></script>
    <script>
      jQuery.noConflict();

      jQuery("<p>I am a paragraph</p>").appendTo(body);

      // $引用 conflictingFramework.js, 而非 jQuery
      $.blahMethodFromOtherLibrary();
    </script>
  </head>
  <body>
  </body>
</html>
```

在源代码中,可以看到\$.noConflict()的实际定义:


```
...
noConflict: function( deep ) {
    window.$ = _$;

    if ( deep ) {
        window.jQuery = _jQuery;
    }

    return jQuery;
},
...
```

可以看到.noConflict()方法非常简短。jQuery 维护着一个对\$值的内部引用——在调用\$.noConflict()之后，\$将用于存储之前的值，即之前加载的可能导致冲突的 JavaScript 库中的值。

3.6 小结

本章介绍了基于 jQuery 的 JavaScript 程序设计的结构和语法，介绍了 jQuery 如何充分利用非侵扰式 JavaScript 模式的优点，将行为从页面的标记中分离出来。本章还非常简要地介绍了 jQuery 核心库。学习 jQuery 源代码，可以让你不再把 jQuery 视为一个“黑盒”，与其他 Web 开发人员相比，你可以从 jQuery 源代码中受益匪浅。研究 jQuery()核心库还有助于开发人员学习一些高级 JavaScript 开发技术，可以将这些技术应用于我们自己的项目之中。

经过本章的学习，我们已经具备了一些 jQuery 的基础知识，第 4 章将深入介绍 jQuery 最强大的特性——非常完备的选择器机制。

3.7 参考

<http://jqfundamentals.com/book/index.html#example-3.13>

<http://paulirish.com/2010/10-things-i-learned-from-the-jquery-source/>

<http://ejohn.org/blog/ultra-chaining-with-jquery/>

<http://perfectionkills.com/instanceof-considered-harmful-or-how-to-write-a-robust-isarray/>

<http://stackoverflow.com/questions/5773723/difference-between-jquery-isplainobject-and-jquery-isemptyobject/5773738#5773738>

<http://stackoverflow.com/questions/122102/what-is-the-most-efficient-way-to-clone-a-javascript-object>

<http://nfriedly.com/techblog/2009/06/advanced-javascript-objects-arrays-and-array-like-objects/>

<http://en.wikipedia.org/wiki/Queue>

http://www.bitstorm.org/weblog/2011-2/How_to_use_the_jQuery_queue_function.html
<http://www.myinkblog.com/2009/09/11/sizzle-a-look-at-jquery-s-new-css-selector-engine/>
<http://www.youtube.com/watch?v=ijpD8ePLBds>
<http://webhole.net/2009/11/28/how-to-read-json-with-javascript/>
<http://ajax.sys-con.com/node/676031>
<http://www.w3.org/DOM/#what>
<http://www.quirksmode.org/dom/intro.html>



第 4 章

选择和操作 DOM 元素

本章内容

- jQuery 选择器的威力
- 遍历 DOM
- 访问和修改 HTML 元素

在第 3 章已经介绍了 jQuery 的核心技术，简要地介绍了 jQuery 的工作原理。本章将更深入地介绍 jQuery 的选择器，以及如何通过 jQuery 选择器选取和操作元素。

用 iPhone 式的格言来说，“让选择器来完成一切”就是 jQuery 的座右铭。对 Web 应用程序开发来说，必须掌握 jQuery 常见的使用模式，即选择一个 DOM 元素，然后对选中的元素执行所需的操作。在 jQuery 选择器引擎的帮助下，选取元素非常简单。如果你曾经使用过 CSS 选择器，那么将会感到 jQuery 选择器非常亲切。jQuery 中的选择器语法结合了 CSS1-3 和 XPath 选择器的语法。

本章将深入介绍 jQuery 选择器的使用方法。可以采用多种方法来获取元素、控制选取元素的上下文并在指定的上下文中选取元素。本章还介绍了如何在运行时生成新的 HTML 代码，另外还将介绍一些用于操作 DOM 的快捷方法。

要想以交互方式试验各种 jQuery 选择器，可以使用交互式的 jQuery 测试程序，比如网站 www.woods.iki.fi/interactive-jquery-tester.html 所提供的测试程序。要使用这个交互式测试程序，只需要在该页面顶部的文本输入框中输入要测试的 jQuery 选择器，并在页面右下方的文本区域中输入供测试的 HTML 代码。随着选择器模式发生改变，该选择器匹配的元素将高亮显示在页面左下方的文本区域中。

4.1 jQuery 选择器的功能

也许在某些情况下，每一个 JavaScript 程序员都努力编写过功能类似的代码。比如查找具有 CSS 类 `highlighted` 的所有表格行，然后将这些表格行的 CSS 类修改为 `normal`，比如下面的代码：

```
var tr = document.getElementsByTagName("tr");
for ( var = i; i < tr.length; i++ ) {
    if ( tr[i].class === 'highlighted' ) {
        tr[i].class = 'normal';
    }
}
```

上面的代码实现了所需的功能，但很繁琐。采用 jQuery 来实现则非常简洁：

```
$("#tr.highlighted").removeClass("highlighted").addClass("normal");
```

无循环语句、无 if 语句、无中间变量，jQuery 将 6 行 JavaScript 代码精简为 1 行 jQuery 代码，并且具有良好的可读性。这非常棒！在上面的代码中，使用了几个不同的 jQuery 概念：一是选择器，二是链式调用，三是工具方法。最值得注意的要点是：jQuery 选择器极大地减少了获得一个 DOM 元素数组所需的代码数量。

4.1.1 选择元素

jQuery 提供了大量方法用于选择 DOM 元素，可以通过元素的属性、元素类型、元素的位置、CSS 类或以上方法的组合来选择 DOM 元素。选择元素的语法如下所示：

```
$(selector, [context])
```

或者

```
jQuery(selector, [context])
```

要根据元素的标记名称来选择元素，只需要将标记名称作为选择器即可。例如，`$("div")` 将选择文档中所有的 `div` 元素。在一个已经加载了 jQuery 库的 `jconsole` 中运行上面的例子，就可以在 Firebug 窗口中看到它返回的对象。

在下面这个例子中，使用了与之相同的 `div` 选择器，在 HTML 文档的 `<body>` 标记中包含了三个空的 `div` 元素。利用 `length` 属性，就可以通过 `console.log` 方法来验证选择器所返回 `div` 元素的数量为 3。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
```

```

$(document).ready(function(){
    var wrappedElements = $("div");
    console.log(wrappedElements.length);
});
</script>
</head>
<body>
    <div id="1"></div>
    <div id="2"></div>
    <div id="3"></div>
</body>
</html>

```

代码片段 tag-selector.txt

另外，还可以通过元素的 ID 属性来选择元素。例如，`$("#div#elementId")` 将选择所有 ID 为 `elementId` 的 `div` 元素。通常情况下，ID 选择器的格式是：元素名、后跟一个井号(#)，紧接着是要匹配的元素 ID。下面的代码示例演示了 ID 选择器的使用方法，它选择了一个 ID 为 `myList` 的元素。



```

<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(document).ready(function(){
        var wrappedElements = $("ul#myList");
        console.log(wrappedElements.length);
      });
    </script>
  <body>
    <div id="main">
      <ul id="myList">
        <li>foo</li>
        <li>bar</li>
      </ul>
    </div>
  </body>
</html>

```

代码片段 id-selector.txt

可以使用星号选择器(即*)来选取 DOM 中所有的元素，比如下面的例子。它返回的选中元素个数为 9。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(document).ready(function(){
        var allElements = $("*");
        console.log(allElements.length);
      });
    </script>
  <body>
    <div id="main">
      <ul id="myList">
        <li>foo</li>
        <li>bar</li>
      </ul>
    </div>
  </body>
</html>
```

代码片段 *all-selector.txt*

默认情况下，当选择元素时 jQuery 将搜索整个 DOM 树。有时，可能只想搜索 DOM 的一个子树。要实现这样的功能，可以向 jQuery() 函数传入第 2 个参数，为搜索指定一个上下文。例如，在文档中具有一系列的链接，但我们只想选取 DOM 中一个指定 div 中的链接，此时可以使用下面的代码：

```
$("#a", "js_links#div");
```

4.1.2 CSS 样式选择器

与 CSS 选择器类似，如果想通过 CSS 类选择元素，只需要使用点号(.)。例如：

```
$("#p.highlighted");
```

到目前为止，我们都使用 \$("element#id")、\$("element.class") 或与之类似的格式来选择元素。实际上，在过滤器之前不必使用元素名称。可以将前面的例子重新改写为下面的代码：

```
$("#stuffId")
$(".myCssClass")
```

由于某些字符——比如井号(#)——具有特殊的含义。如果想在选择器表达式中使用特殊字符，应该使用反斜线对这些特殊字符进行转义。对于下面的特殊字符列表，其中每一个字符在选择器中都具有特殊含义，如果想匹配这些特殊字符，应该使用反斜线进行转义：

```
# ; & , . + * ~ ' : " ! ^ $ [ ] ( ) = > | /
```

例如，如果在某个 HTML 文档中使用美元符号作为命名规范，则应该在 ID 选择器中使用反斜线对其进行转义，比如下面的代码：



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      var n = $("#\\$specialId").length;
      console.log(n);
    </script>
  </head>
  <body>
    <div id="$specialId">T-Rex</div>
  </body>
</html>
```

代码片段 *escaped-selectors.txt*

在这一小节中还要介绍两个基本的选择器：复式选择器和后代选择器。后代选择器的格式是 `$("ancestor descendant")`，它返回一个包装对象，其中包含了指定祖先元素的所有后代元素。比如：

```
$("form input")
```

它将返回一个包装对象，其中包含了表单的所有后代元素。

在 `$()` 函数的参数中，没有任何限制只能传入一个选择器表达式。可以组合不同种类的选择器，形成一个以逗号分隔的选择器列表：

```
$("div#gallery, div#userInfo, form");
```

表 4-1 总结了 jQuery 基本选择器的类型。

表 4-1 基本选择器

选 择 器	描 述
*	通配符选择器
.	CSS 类选择器
"Element"	元素选择器
#	ID 选择器
"select1, select2"等	复式选择器

4.1.3 属性选择器

jQuery 可以根据元素的属性，以各种非常灵活的方式来选择元素。最简单的属性选择器就是与某个字符串模式精确匹配：

```
$("#[attributeName='string2match']")
```

或者

```
jQuery("[attributeName='string2match']")
```

一些属性选择器采用了正则表达式的表示法，可以匹配字符串中的一部分。例如匹配属性的开头，或者匹配属性的结尾。属性开头选择器用于选择属性值以某个特定字符串开头的元素。属性结尾选择器则用于选择属性值以某个特定字符串结尾的所有元素：

```
$("#[attributeName^='value']"); //匹配属性开头
```

```
$("#[attributeName$='value']"); //匹配属性结尾
```

假如页面中具有一些 ID 以字符串'user'开头的<input>元素，下面的代码示例演示了如何使用一个正则表达式来选择所有这些 ID 以字符串'user'开头的元素：



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>

      var userInfo = $("#[id^='user']").length;
      console.log(n);

    </script>
  </head>
  <body>
    <form>
      <input id="userName" type="text" />
      <input id="userId" type="text" />
      <input id="userPhone" type="text" />
    </form>
  </body>
</html>
```

代码片段 `attribute-starts-selector.txt`

与匹配的属性选择器相比，还可以使用不匹配的属性选择器。下面的例子演示了如何使用不匹配的属性选择器，选择 ID 不等于'cheese'的元素。



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>

    <script>
      var userInfo = $("[id!='cheese']").length;
      console.log(console.log(n));
    </script>

  </head>
  <body>
    <div id="eggs"></div>
    <div id="ham"></div>
    <div id="cheese"></div>
  </body>
</html>
```

代码片段 *negative-selector.txt*

另外，还可以使用*=通配符，选择属性值中任意位置包含指定子串的元素：

```
$("[attributeName*='value']");
```

假如在某个文档中具有一组 div 元素，每一个 div 元素都具有一个 ID 属性，我们想选择那些 ID 属性中包含字符串'zombie'的 div 元素子集，其中'zombie'可以出现在 ID 属性值的开头或结尾。下面的代码演示了如何选择在 ID 属性值的任意位置包含字符串'zombie'的所有 div 元素：



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>

    <script>
      var userInfo = $("[id*='zombie']").length;
      console.log(n);
    </script>

  </head>
  <body>
    <div id="greenzombie"></div>
    <div id="zombieman"></div>
    <div id="madscientist"></div>
  </body>
</html>
```

代码片段 *anywhere-selector.txt*

经过上面的学习我们已经知道可以用多种方式来选择元素，比如精确匹配属性的值、与属性值的开头或结尾匹配、在属性值中不包含指定的字符串、在属性值中任意位置包含特定的字符串。除此之外，jQuery 还有更多选择元素的方法。比如值列表中包含某个特定值的选择器，当属性值是一个以空格分隔的值列表时，它用于选择属性值列表中包含一个特定值的元素。下面代码使用了这种选择器。

```
$("#[attributeName~='value']");
```

对于从一个输入框中抽取值时这非常有用(或者其他很多场合)。例如，下面的代码搜索整个 DOM 树，选择 ID 属性值中包含单词 zombie 的元素。



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>

    <script>
      var userInfo = $("#[id~='zombie']").length;
      console.log(n);
    </script>

  </head>
  <body>
    <div id="greenzombie"></div>
    <div id="zombie man"></div>
    <div id="mad scientist"></div>
  </body>
</html>
```

代码片段 attribute-contains.txt

与基本选择器类似，对于更加复杂的情形还可以使用复式属性选择器。在下面的例子中定义了 4 个表单：其中两个表单包含了 admin 用户的数据，另外两个表单包含了普通用户的数据。这 4 个表单中的每一个表单都分别包含了三个文本输入框：userName、employeeId 和 age。表单的名称采用如下所示的命名规范：userType/User/OfficeNumber。要选择所需的表单，比如选取 adminUserOffice1 表单，可以使用复式属性选择器 `$("#form[name$='Office1'] form[name^='admin']")`，或者写为 `$("#[name$='Office1'] [name^='admin']")`。另外，也可以使用 name 属性直接匹配表单名称字符串，比如 `$("#[name='adminUserOffice1']")`。下面的例子演示了复式属性选择器的使用。



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
```

```

</script>
<script>
    $(function(){
        var n = $("form[name$='Office1'] form[name^=admin]").length;
        console.log(n);
    });
</script>
</head>
<body>
    <form name="adminUserOffice1" method="post" action="sendUserData.php">
        <input type="text" id="userName"></input>
        <input type="text" id="employeeId"></input>
        <input type="text" id="age"></input>
    </form>
    <form name="adminUserOffice2" method="post" action="sendUserData.php">
        <input type="text" id="userName"></input>
        <input type="text" id="employeeId"></input>
        <input type="text" id="age"></input>
    </form>
    <form name="regularUserOffice1" method="post" action="sendUserData.php">
        <input type="text" id="userName"></input>
        <input type="text" id="employeeId"></input>
        <input type="text" id="age"></input>
    </form>
    <form name="regularUserOffice2" method="post" action="sendUserData.php">
        <input type="text" id="userName"></input>
        <input type="text" id="employeeId"></input>
        <input type="text" id="age"></input>
    </form>
</body>
</html>

```

代码片段 *multiple-attribute-selectors.txt*

表 4-2 总结了不同种类的属性选择器。

表 4-2 属性选择器

属性选择器	描 述
<code>elem[attr]</code>	选择具有 attr 属性的元素
<code>elem[attr=val]</code>	选择具有 attr 属性且属性值与 val 值匹配的元素
<code>elem[attr^=val]</code>	选择具有 attr 属性且属性值以 val 值开头的元素
<code>elem[attr =val]</code>	选择具有 attr 属性且属性值以 val 值开头或者属性值等于 val 值的元素
<code>elem[attr\$=val]</code>	选择具有 attr 属性且属性值以 val 值结尾的元素
<code>elem[attr!=val]</code>	选择具有 attr 属性且属性值不等于 val 值的元素
<code>elem[attr~val]</code>	选择具有 attr 属性，且属性值是一个以空格分隔的列表，其中包含 val 值的元素
<code>elem[attr*=val]</code>	选择具有特定的 attr 属性，且属性值中含有一个指定子串的元素

4.1.4 位置选择器

jQuery 还可以根据元素相对与其他元素的位置(子/父关系)来选择元素, 或者根据元素在文档中的层次关系来选择元素。比如选择匹配集合中的第一个或最后一个元素, 或者选择表格中的偶数行。在下面的代码块中, 包含了一个有序列表, 该列表用一个 id 为 fowl 的元素来表示。随后的 JavaScript 代码演示了多种 jQuery 位置选择器。这些位置选择器包括:

- li:even 返回匹配集合中的偶数成员
- li:odd 返回匹配集合中的奇数成员
- li:first 返回匹配集合中的第一个元素
- li:last 返回匹配集合中的最后一个元素
- li:eq(3)返回匹配集合中的第 4 个元素
- li:gt(2)返回匹配集合中的索引值大于 2 的所有元素
- li:lt(3)返回匹配集合中的索引值小于 3 的所有元素

请注意, 除了 nth-child 选择器的索引是从 1 开始之外, 所有位置选择器的索引都是从 0 开始的。在“关系过滤选择器”小节中将介绍 nth-child 选择器。

```
<ol id="fowl">
  <li>Turkey</li> <!-- 0 -->
  <li>Chicken</li> <!-- 1 -->
  <li>Parrot</li> <!-- 2 -->
  <li>Pigeon</li> <!-- 3 -->
  <li>Hawk</li> <!-- 4 -->
</ol>
$("li:even") //返回 turkey、parrot 和 hawk
$("li:odd") //返回 chicken 和 pigeon
$("li:first") //返回 turkey
$("li:last") //返回 hawk
$("li:eq(3)") //返回 pigeon
$("li:gt(2)") //返回 pigeon 和 hawk
$("li:lt(3)") //返回 parrot、chicken、turkey
```

4.1.5 过滤选择器

请注意, 上面的选择器全都以一个分号(:)开始。这一类选择器称为过滤选择器, 因为它们的功能是对基本选择器进行过滤。jQuery 添加了更多的过滤器, 有专门用于表单元素的过滤器, 比如:button 或:input; 检查元素是否处于动画状态的过滤器和其他的一些过滤器。在 jQuery 的文档中, 将这些过滤器分类为基本过滤器、可见性过滤器、内容过滤器、子元素过滤器和表单过滤器。其中, :animated 过滤器将在第 7 章中进行介绍。

1. 基本过滤选择器

表 4-3 列出了 jQuery 中的基本过滤器。它们的使用方法与前面的例子类似。

表 4-3 基本过滤选择器

类 型	描 述
:animated	选择当前正在执行动画的所有元素
:eq()	选择索引等于指定值的元素
:even	选择索引值为偶数的所有元素
:first	选择第一个元素
:gt()	选择索引大于指定值的元素
:header	选择所有的标题元素, 比如 h1、h2、h3 等
:last	选择最后一个元素
:lt()	选择索引小于指定值的元素
:not()	选择与选择器不匹配的元素
:odd	选择索引值为奇数的所有元素

2. 过滤表单元素

大家都知道, 验证、解析和提交表单数据是 Web 开发人员每一天都要面对的工作。因此 jQuery 提供了一些过滤器, 使选择表单元素的琐碎工作变得更加轻松。这里所说的表单元素, 指的是属于一个 HTML 表单的特定元素, 比如用于将数据提交给服务器的 button 元素, 以及 input 元素和 textarea 元素等。



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        var n = $(':input').length;
        console.log(n);
      });
    </script>
  </head>
  <body>
    <form name="" method="post">
      <input type="text" />
      <input type="hidden" />
      <input type="password" />
      <input type="radio" />
      <input type="reset" />
      <input type="submit" />
      <input type="image" />
      <input type="file" />
      <input type="checkbox" />
    </form>
  </body>
</html>
```



```



```

代码片段 *filtering-forms.txt*

:input 选择器用于选择所有的表单控件，包括<select>、<textarea>和<button>。另外，还有用于筛选特定表单控件的更具体的过滤器，比如:button、:password 和:text。从这些过滤器可以看到，jQuery 在如何选择元素的问题上是非常灵活的，可以采用不止一种方法来获得相同的选择结果。下面的代码片段演示了选择同一表单中所有元素的各种不同方法。



```

<!DOCTYPE html>
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.7.1.js">
  </script>
  <script>
    $(function(){
      var n1 = $("input").length;
      var n2 = $(":input").length;
      var n3 = $("form > *").length;
      var n4 = $(":text").length;
      var n5 = $("input[type='text']").length;
      console.log(n1 + ", " + n2 + ", " + n3 + ", " + n4 + ", " + n5);
    });
  </script>
</head>
<body>
  <form name="" method="post">
    <input type="text" />
    <input type="text" />
    <input type="text" />
  </form>
</body>
</html>

```

代码片段 *multiple-form-filters.txt*

如果表单中具有<textarea>或<button>元素，\$("form > *")和\$(":input")也将返回这些元素。表 4-4 总结了所有种类的表单元素过滤器。

表 4-4 表单元素过滤器

过 滤 器	描 述
:button	选择所有 button 元素和类型为 button 的元素
:checkbox	选择所有类型为 checkbox 的元素
:checked	匹配所有已被选中的元素
:disabled	选择所有不可用元素
:enabled	选择所有可用元素
:file	选择所有类型为 file 的元素
:image	选择所有类型为 image 的元素
:input	选择所有 input、textarea、select 和 button 元素
:password	选择所有类型为 password 的元素
:radio	选择所有类型为 radio 的元素
:reset	选择所有类型为 reset 的元素
:selected	选择所有已选中元素
:submit	选择所有类型为 submit 的元素
:text	选择所有类型为 text 的元素

3. 可见性过滤器

jQuery 根据元素的 offsetWidth 和 offsetHeight 属性来判断一个元素是否可见。如果这两个属性都是 0，则 jQuery 认为该元素是隐藏的。如果直接将元素的样式设置从 display:block;修改为 display:none;——通过这种方式来切换元素的可见性，jQuery 将检测到元素这些可见/不可见状态的改变。可以使用:hidden 和:visible 过滤器来匹配元素的可见性。下面的例子演示了:hidden 和:visible 过滤器的使用方法。



```
<!DOCTYPE html>
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.7.1.js">
  </script>
  <script>
    $(function(){
      var numInv = $(":text:hidden").length;
      var numVis = $(":text:visible").length;
      console.log(numInv);
      console.log(numVis);
    });
  </script>
```

```

</head>
<body>
  <form name="" method="post">
    <input type="text" name="text1" style="display:none;"/>
    <input type="text" name="text2" style="offsetWidth:0; offsetHeight:0;"/>
    <input type="text" name="text3" style="display:block;"/>
  </form>
</body>
</html>

```

代码片段 visibility-filters.txt

表 4-5 可见性过滤器

类 型	描 述
:hidden 选择器	选择所有隐藏元素
:visible 选择器	选择所有可见元素

4. 内容过滤器

在某些情况下，需要根据一个元素包含(或者不包含)某种内容进行选择，比如包含某些文本。使用:contains()过滤器，可以选择在元素直接的内容中、元素的后代中、或者同时在二者中包含匹配文本的元素。下面的例子演示了:contains()过滤器的使用，它选择了所有包含文本字符串 jenny 的<p>元素，并将其保存到一个名为 jennies 的变量中。



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        var jennies = $("p:contains('jenny')").length;
        console.log(jennies); //返回值 2
      });
    </script>
  </head>
  <body>
    <p>jenny smith</p>
    <p>jennyjones</p>
    <p>jim bob</p>
  </body>
</html>

```

代码片段 content-filter.txt

另外一个与内容过滤有关的过滤器是:has(), 它返回这样的元素: 在该元素的后代元素中至少包含一个与指定选择器匹配的元素。:has()过滤器可用于选择包含与指定选择器匹配的后代元素的元素。它并不局限于与直接子元素进行匹配。下面的代码示例演示了在页面上所有 div 元素中, 如何使用:has()过滤器来选择其后代元素中包含一个<p>元素的 div 元素:



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function() {
        var hasDemo = $("div:has('p')").attr("id");
        console.log(hasDemo); //返回'yay'
      });
    </script>
  </head>
  <body>
    <div id="yay">
      <ul>
        <li>
          <p>The glorious paragraph</p>
          <p>Another, glorious paragraph</p>
          <p>The best paragraph yet</p>
        </li>
      </ul>
    </div>
    <div id="nay">
      <ul>
        <li> No paragraphs here. </li>
      </ul>
    </div>
  </body>
</html>
```

代码片段 has-selector.txt

如果想选择不包含任何内容的空元素, 可以使用:empty()选择器。该选择器返回所有无子元素或文本的元素。



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function() {
        var nothing = $("p:empty").length;
        console.log(nothing) //返回值为 1
      });
    </script>
  </head>
  <body>
    <p></p>
  </body>
</html>
```



```

    </script>
</head>
<body>
    <div>
        <p></p>
        <p>something here</p>
    </div>
</body>
</html>

```

代码片段 *empty-selector.txt*

与:empty()相对的是:parent()选择器。它的功能是选择具有子元素的元素。下面的代码示例演示了:parent()的使用,它从文档中选择一个具有子元素的 div 元素,并输出该 div 元素的 ID:



```

<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function() {
        var parentDemo = $("div:parent");
        console.log(parentDemo.attr("id")) //返回 proudParent
      });
    </script>
  </head>
  <body>
    <div id="empty"></div>
    <div id="proudParent">
      <ul>
        <li> Children! </li>
      </ul>
    </div>
  </body>
</html>

```

代码片段 *parent-selector.txt*

值得注意的是,:parent()也会选择包含子文本节点的元素。子文本节点可能只包含空格字符,比如在下方的例子中,id 为#empty 的 div 元素并非真正的空元素,它包含了一个空格字符。因此它将与:parent()选择器匹配。



```

<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">

```

```

</script>
<script>
    $(function(){
        var parents = $("div:parent").length;
        console.log(parents) //返回 2
    });
</script>
</head>
<body>
    <div id="empty">
    </div>
    <div id="proudParent">
        <ul>
            <li> Children! </li>
        </ul>
    </div>
</body>
</html>

```

代码片段 whitespace-parent.txt

表 4-6 列出了 jQuery 中可用的内容过滤器。

表 4-6 内容过滤器

类 型	描 述
:contains()	选择所有包含特定文本内容的元素
:empty	选择所有不包含子元素或文本的空元素
:has()	选择至少含有一个元素与指定选择器相匹配的元素
:parent	选择所有含有子元素或文本节点的元素

5. 根据关系进行过滤

下面的代码示例演示了如何使用过滤选择器，与特定的关系进行匹配。例如：first-child 或:only-child。



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        console.log( $("div span:first-child") )
        //[span#turkey, span#bear, span#martian]
        console.log( $("div span:last-child") )
        //[span#hawk, span#horse, span#martian]
      });
    </script>
  </head>
  <body>
    <div>
      <span id="turkey"> Turkey
      <span id="bear"> Bear
      <span id="martian"> Martian
    </div>
    <div>
      <span id="hawk"> Hawk
      <span id="horse"> Horse
      <span id="martian"> Martian
    </div>
  </body>
</html>

```

```

        console.log( $("div span:only-child") )
        //[span#martian]
        console.log( $("div span:nth-child(2)") )
        //[span#chicken, span#rabbit]
        console.log( $("div span:nth-child(2n+1)") )
        //[span#turkey, span#parrot, span#hawk, span#bear, span#fox,
        //span#horse, span#martian]
        console.log( $("div span:nth-child(even)") )
        //[span#chicken, span#pigeon, span#rabbit, span#monkey]
    });
</script>
</head>
<body>
    <div>
        <span id="turkey">Turkey</span>
        <span id="chicken">Chicken</span>
        <span id="parrot">Parrot</span>
        <span id="pigeon">Pigeon</span>
        <span id="hawk">Hawk</span>
    </div>
    <div>
        <span id="bear">bear</span>
        <span id="rabbit">rabbit</span>
        <span id="fox">fox</span>
        <span id="monkey">monkey</span>
        <span id="horse">horse</span>
    </div>
    <div>
        <span id="martian">martian</span>
    </div>
</body>
</html>

```

代码片段 relationship-filter.txt

表 4-7 总结了这一小节介绍的位置选择器。

表 4-7 子元素过滤器

选 择 器	描 述
:first-child	选择每个父元素的第一个子元素
:last-child	选择每个父元素的最后一个子元素
:nth-child()	选择每个父元素的第 nth-child() 个子元素
:only-child	选择具有唯一一个子元素的元素

4.1.6 用户自定义选择器

如果当前 jQuery 内置的选择器不够用,开发人员也可以扩展 jQuery,实现用户自定义的选择器。要创建自定义的选择器,必须扩展 jQuery 对象。下面的代码示例创建了一个用户自定义的选择器,用于选择具有绿色背景的元素:



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        // 通过扩展$.expr[":"]实现自定义选择器
        $.expr[":"].greenbg = function(element) {
          return $(element).css("background-color") === "green";
        };
        var n = $(":greenbg").length;
        console.log("There are " + n + " green divs");
      });
    </script>
  </head>
  <body>
    <div style="width:10; height:10; background-color:green;"></div>
    <div style="width:10; height:10; background-color:black;"></div>
    <div style="width:10; height:10; background-color:blue;"></div>
  </body>
</html>
```

代码片段 custom-fi lters.txt

支持创建用户自定义的高级选择器是一项非常受欢迎的功能,开发人员不必再受限于 jQuery 内置的选择器。

4.2 遍历 DOM

经过上面的学习,相信你已经对 jQuery 选择器的强大功能深信不疑。jQuery 选择器提供了各种各样的选择功能,它从 DOM 中选择元素并创建包装对象。在 jQuery 的“武器库”中,还需要管理包装对象集合的功能。比如处理诸如获取元素引用、获取原集合子集的一个包装对象、或者常见的遍历 DOM 等, jQuery 可以轻松处理这一切。

使用 `.find()` 方法可以从一个元素包装集中查找与指定选择器匹配的后代元素,它的使用方式类似于使用 jQuery 函数进行元素的选择。`.find()` 方法可以接收一个选择器字符串。使用 `.find()` 方法的实际效果,类似于对当前正在操作的包装集进行更新操作。与 jQuery 函数相比,在使用 `.find()` 方法时必须传入一个选择器表达式作为参数。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        var wrapper = $("ul");
        console.log("selected list count:" + wrapper.length);
        var newWrapper = wrapper.find("#techBooks");
        console.log("selected list count:" + newWrapper.length);
      });
    </script>
  </head>
  <body>
    <ul id="groceries">
      <li>Eggs</li>
      <li>Bacon</li>
      <li>Ham</li>
    </ul>
    <ul id="books">
      <li>
        <ul id="techBooks">
          <li>Pro jQuery</li>
          <li>JavaScript 101</li>
        </ul>
        <ul id="mathBooks">
          <li>Calculus for the Gods</li>
          <li>Intro to Finite State Machines</li>
        </ul>
      </li>
      <li>How to Win at Jeopardy</li>
      <li>How To Cook</li>
    </ul>
  </body>
</html>
```

代码片段 find.txt

在上面的例子中，第一个 `console.log` 输出值为 4，而第二个 `console.log` 的输出值为 1。`.find()` 方法将搜索所有的后代元素，如果只想在第一层后代元素中搜索，则应该使用 `.children()` 方法。

jQuery 还提供了一些快捷方式，用于访问元素包装集中特殊位置的元素，比如最常见的情况——获取一个包装集中的第一个元素或者最后一个元素。jQuery 正好提供了这种快捷方式，比如下面的例子：



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        var listElements = $("li");
        var firstEl = listElements.first();
        var lastEl = listElements.last();

        console.log("firstEl value:" + firstEl.html());
        console.log("firstEl length:" + firstEl.length);

        console.log("lastEl value:" + lastEl.html());
        console.log("lastEl length:" + lastEl.length);
      });
    </script>
  </head>
  <body>
    <ul>
      <li>eggs</li>
      <li>bacon</li>
      <li>ham</li>
    </ul>
  </body>
</html>
```

代码片段 children.txt

上面例子中的页面具有一个无序列表，它包含了三个列表项：eggs、bacon 和 ham。首先使用 jQuery 函数选择了所有三个列表项，然后再分别使用 `.first()` 和 `.last()` 方法，获取第一个和最后一个包装对象。为了检验选择结果，调用了两个 `console.log` 方法，分别用于显示 `.first()` 和 `.last()` 方法所获取包装对象的长度，这两个 `console.log` 的输出值都应该是 1。你可能注意到在上面的代码中调用了 `.html()` 方法，该方法的功能是获取列表项的内容，本章后面将介绍 `.html()` 方法的使用。

还可以使用 `.eq()` 方法，获取包装集中特定索引位置的元素。与 `.first()` 和 `.last()` 方法类似，`.eq()` 方法的返回值也是一个包装对象。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        var listElements = $("li");
```



```

        var secondEl = listElements.eq(1);
        var fourthEl = listElements.eq(3);
        console.log(secondEl.html() + " and " + fourthEl.html());
    });
</script>
</head>
<body>
<ul>
    <li>eggs</li>
    <li>bacon</li>
    <li>ham</li>
    <li>cheese</li>
    <li>juice</li>
    <li>sausage</li>
</ul>
</body>
</html>

```

代码片段 eq.txt

console.log 将输出 bacon and cheese。

在某些情况下，我们可能想直接获得一个 DOM 节点。对于这种情况可以使用.get()方法，该方法接收一个索引值作为参数。get()方法中的索引值是基于 0 的，即索引是从 0 而不是 1 开始计数的。在下面的代码示例中，描述了如何使用.get()方法访问匹配集成员底层的 DOM 元素。



在 jQuery 中还有另外一个.get()方法，它用于 Ajax 请求。可以从使用方式上来区分这两个方法。Ajax 的.get()方法是直接从 jQuery 函数调用的(\$.get())，而上面讨论的获取 DOM 元素的.get()方法，是从一个 jQuery 选择器生成的匹配集上调用的。



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
<head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
        $(function() {
            var listElements = $("li");
            var secondEl = listElements.get(2);
            var fourthEl = listElements.get(3);
            console.log(secondEl.innerHTML + " and " + fourthEl.innerHTML);
            // "ham and cheese"
        });
    </script>
</head>
<body>
<ul>
    <li>eggs</li>
    <li>bacon</li>
    <li>ham</li>
    <li>cheese</li>
    <li>juice</li>
    <li>sausage</li>
</ul>
</body>
</html>

```

```

    </script>
</head>
<body>
  <ul>
    <li>eggs</li>
    <li>bacon</li>
    <li>ham</li>
    <li>cheese</li>
    <li>juice</li>
    <li>sausage</li>
  </ul>
</body>
</html>

```

代码片段 get.txt

.get()方法接收一个索引值作为参数,并返回一个 DOM 节点。另外,还可以使用.index()方法执行与之相反的操作。该方法接收一个 DOM 元素作为参数,返回该 DOM 元素在匹配集中的索引值。与.get()方法一样,.index()方法的索引值也是基于 0 的。



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        var item2 = document.getElementById("item2");
        console.log($("#li").index(item2));
      });
    </script>
  </head>
  <body>
    <ul>
      <li id="item0">eggs</li>
      <li id="item1">bacon</li>
      <li id="item2">ham</li>
      <li id="item3">cheese</li>
      <li id="item4">juice</li>
      <li id="item5">sausage</li>
    </ul>
  </body>
</html>

```

代码片段 index.txt

对于.index()方法,不仅可以传入一个 DOM 元素作为参数,还可以传入一个 jQuery 包装对象作为参数。如果传入的包装对象包含了一个以上的元素,则.index()方法返回第

一个匹配元素的索引值。

.index()方法还可以接收第三种类型的参数，即传入一个选择器字符串作为参数。如果该选择器返回的元素不止一个，则.index()方法返回第一个与之匹配的元素索引值，该索引值是基于0的。.get()和.index()这两个方法都被视为是“解构性”方法，因为它们中断了jQuery的调用链。当一个方法的调用并不返回一个jQuery包装对象时，就会造成jQuery调用链的中断，无法继续链接调用其他jQuery方法。

可以将一个调用链视为一个调用栈，栈中的每一项都是一个元素集合。当调用一个方法时，比如调用.filter()或者.find()方法，一个新的元素集将被压入调用栈中。有时我们可能想返回到最初的包装对象，此时可以使用.end()方法来实现，比如下面的代码示例：



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        var listElements = $("li");
        listElements
          .filter(":gt(3)")
          .css("color", "green")
        .end()
          .filter(":lt(3)")
          .css("color", "red")
        .end();
        console.log( listElements.length );
      });
    </script>
  </head>
  <body>
    <ul>
      <li>eggs</li>
      <li>bacon</li>
      <li>ham</li>
      <li>cheese</li>
      <li>juice</li>
      <li>sausage</li>
    </ul>
  </body>
</html>
```

代码片段 end.txt

在上面的代码中，首先在初始包装对象 listElements 上调用了.filter()方法，选择了最后两个列表项，然后使用.css()方法将文本颜色修改为绿色。此外我们还想把每一个索引值小于3的列表项的颜色修改为红色，因此调用了.end()方法，使当前操作的包装对象返回到初

始列表项的集合。请注意，在上面代码的调用链中使用了一些格式上的缩进，以便代码阅读起来更直观。

到目前为止，本章所有例子都使用了 `.length` 属性来返回选中元素的数量，但实际上我们也可以使用 `.size()` 方法。调用 `.size()` 方法在性能上要比使用 `length` 属性略差，但如果方法调用与其余代码在形式上更加统一，则也可以使用 `.size()` 方法。

有时，我们需要将匹配的元素集转换为一个普通的 JavaScript 数组(Array)。jQuery 提供了一个名为 `.toArray()` 的方法来实现该功能。下面的代码示例演示了如何在实际的应用中使用 `.toArray()` 方法。首先，使用 jQuery 选择了所有列表项(即姓名)，然后将 jQuery 选择器返回的包装对象转换为一个原生的 JavaScript 数组，再使用 Array 对象的 `reverse()` 方法，颠倒数组中元素的顺序。使用循环语句遍历新数组，从输出的列表姓名可以看到代码执行的结果是正确的。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script >
      $(function(){
        var winners = $( "#winners li" ).toArray();
        winners = winners.reverse();
        for ( var i = 0, test = winners.length; i < test; i++) {
          console.log(winners[i].innerHTML);
          /*输出
          Nicolas Frantz
          Mark Cavendish
          André Darrigade
          Lance Armstrong
          André Leducq
          Bernard Hinault
          Eddy Merckx
          */
        }
      });
    </script>
  </head>
  <body>
    <ol id="winners" >
      <li>Eddy Merckx</li>
      <li>Bernard Hinault</li>
      <li>André Leducq</li>
      <li>Lance Armstrong</li>
      <li>André Darrigade</li>
      <li>Mark Cavendish</li>
      <li>Nicolas Frantz</li>
    </ol>
```



```

    </body>
</html>

```

代码片段 toArray.txt

在介绍修改元素内容的 jQuery 方法之前, 让我们最后再介绍一个选择元素的方法 `.andSelf()`。开发人员并不会经常使用到该方法, 但该方法确有重要的价值。在学习 `.end()` 方法时已经介绍过, jQuery 维护着一个栈, 该栈跟踪着匹配元素集的变化。当调用了一个遍历 DOM 的方法, 比如调用了 `.filter()` 或 `.find()` 方法时, 新的匹配集将被压入栈中。如果想同时把之前的匹配集与新匹配集一起也压入栈, 则可以使用 `.andSelf()` 方法进行补救。在下面的代码示例中, 在执行了一个 `.find()` 栈操作后, 使用 `.andSelf()` 方法将之前匹配的 `div` 元素也压入栈中。在新的匹配元素集中, 元素的顺序与它们在 DOM 中的顺序一致。



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
  <head>
    <script src='http://code.jquery.com/jquery-1.7.1.js'>
    </script>
    <script >
      $(function(){
        console.log( $("div") );
        //[ div ]
        console.log( $( "div" ).find("p") );
        //[ p ,p, p ]
        console.log( $( "div" ).find("p").andSelf() );
        //[ div, p, p, p ]
      });
    </script>
  </head>
  <body>
    <div>
      <p>Paragraph</p>
      <p>Paragraph</p>
      <p>Paragraph</p>
    </div>
  </body>
</html>

```

代码片段 andSelf.txt

4.3 访问并修改元素、属性和内容

前面已经介绍了很多 jQuery 选择元素的办法, 有时我们需要对选取的元素执行某种操作, 比如获取或设置元素的文本内容、元素的属性、甚至是 HTML 内容。jQuery 提供了一

些便利的方法以实现这些功能。

4.3.1 操作内容

JavaScript 最值得称道的特性之一，就是可以在运行时修改 Web 页面的内容，不必等待重新加载整个页面。在 jQuery 中，不仅可以很容易地修改节点的内容，而且获取节点内容也很简单。`.text()`方法就是实现这一功能的方法之一。但是请注意，`.text()`方法将把匹配集合中所有元素的文本合并在一起。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script type="text/javascript">
      $(function(){
        var content = $("p").text();
        console.log(content);
      });
    </script>
  </head>
  <body>
    <p>Where's </p>
    <p>The </p>
    <p>Beef?</p>
  </body>
</html>
```

代码片段 text.txt

上面的例子将在控制台中输出“Where’s the beef?”。这一切都运转得很好。如何更新元素的内容呢？也可以采用相同的`.text()`方法，只需要在调用`.text()`方法时传入一个字符串作为参数即可，该字符串包含了想要插入的新文本——无须传入一个参数列表。在下面的例子中，在页面加载之后，`<p>`元素并不会显示文本“blah”，而是显示“The trees are green”。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        $("p").text("The trees are green.");
      });
    </script>
  </head>
  <body>
```



```

    <p>Blah</p>
  </body>
</html>

```

代码片段 text-setter.txt

要获取或设置元素的内容，第二种方式就是使用.html()方法。它的用法与.text()方法一样：当调用不带参数的.html()方法时，获取元素的内容；当调用带参数的.html()方法时，设置元素的内容。那么.text()方法与.html()方法二者有什么区别呢？区别之一就是.text()方法既可用于 XML 文档，也可用于 HTML 文档，而.html()方法只能用于 HTML 文档。区别之二就是.text()方法将获取所有后代元素的文本内容，而.html()方法仅仅获取匹配的元素的文本内容。

在下面的例子中，获取并设置了一个 div 标记的 HTML 内容：



```

<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        var content = $("div").html();
        console.log(content);
        $("div").html("<p style='color:red;'>RED</p>");
      });
    </script>
  </head>
  <body>
    <div>
      <ul>
      </ul>
    </div>
  </body>
</html>

```

代码片段 html.txt

加载该示例页面时，在 Firefox 控制台中可以看到输出结果：

```

<ul>
</ul>

```

在按下回车键之后，该 div 元素的内容将被更替换为一个具有红色文本 RED 的<p>元素。在这个例子中，CSS 样式是硬编码的，但这并不是实现这一效果的最佳方式。最好使用下一小节将要介绍的.css()方法来实现这样的功能。

最后，.text()和.html()这两方法都可以接收一个函数作为参数，以设置元素的内容。这

样一来,开发人员就可以动态创建元素的内容,而并非只能使用静态的文本,比如下面的例子:



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js">
    </script>
    <script>
      $(function(){
        $("div#testHTML").html(function(){
          var content = "";
          for(i = 1; i <=3; i++){
            content += "testing " + i + "...<br />";
          }
          return content;
        });
      });
    </script>
  </head>
  <body>
    <div id="testHTML">
    </div>
  </body>
</html>
```

代码片段 `html-with-function-argument.txt`

在上面的例子中, `<body>` 包含了唯一一个 `<div>` 元素。在选择了 `id` 为 `testHTML` 的 `div` 元素之后,调用了 `.html()` 方法并传入一个匿名函数作为该方法的参数,该匿名函数生成了如下所示的 HTML:

```
testing 1...<br />
testing 2...<br />
testing 3...<br />
```

1. 操作属性

这里所说的属性指的是元素的属性,请不要与 JavaScript 对象的属性相混淆。可以使用 `.attr()` 方法来获取或修改元素的属性。如果在调用 `.attr()` 方法时,传入了两个参数,第一个参数是属性名,第二个参数是属性值,那么该方法将设置元素的一个属性。如果在调用 `.attr()` 方法时只传入了一个属性名作为参数,则 `.attr()` 方法将获取该属性的值。下面的例子演示了如何使用 `.attr()` 方法获取和设置元素的属性:



```
<!DOCTYPE html>
<html>
  <head>
```

```

<style type="text/css">
    .blue {
        Background-color: blue;
    }
</style>
<script src="http://code.jquery.com/jquery-1.7.1.js">
</script>
<script>
    $(function(){
        $("#i_am_blue").attr("class","blue");
        var attribute = $("#i_am_blue").attr();
        console.log(attribute);
    });
</script>
</head>
<body>
    <p id="i_am_blue">
        I am Blue.
    </p>
</body>
</html>

```

代码片段 *attribute-manipulation.txt*

有时不再需要一个属性。在这种情况下，可以使用一个名为 `.removeAttr()` 的方法来移除一个属性。下面的代码示例演示了如何使用 `.removeAttr()` 方法，从 ID 为 `#i_am_not_blue` 的 `<p>` 元素中移除 `class` 属性。



```

<!DOCTYPE html>
<html>
    <head>
        <style type="text/css">
            .blue {
                background-color: blue;
            }
        </style>
        <script src="http://code.jquery.com/jquery-1.7.1.js">
        </script>
        <script>
            $(function(){
                $("#i_am_not_blue").removeAttr("class");
            });
        </script>
    </head>
    <body>
        <p id="i_am_not_blue" class="blue">
            I am not Blue.
        </p>
    </body>

```



```
</html>
```

代码片段 *removeAttr.txt*

对于这个例子，不必使用 `console.log` 来验证结果。在加载该示例页面之后，可以看到段落中的文本是黑色的。

4.4 生成 HTML

此外，使用 jQuery 的包装函数也可以生成代码。只需要将表示 HTML 代码的字符串作为参数，传递给 jQuery 包装函数即可，比如下面的例子：

```
$("<p>My nifty paragraph</p>");
```

但这仅仅完成了一半的工作，因为新创建的 `<p>` 元素还未添加到 DOM 中相应的父元素中。还需要使用一个类似于 `.appendTo()` 的 jQuery 方法，将新创建的 `<p>` 元素插入到 DOM 中。生成 HTML 标记代码与选择器和链式调用相结合，可以创建出非常简洁的 jQuery 语句。下面的代码示例演示了 HTML 标记代码的生成、结合 `.appendTo()` 方法和 `.attr()` 方法，将一个新的 `<H1>` 元素插入到页面中。



```
<!DOCTYPE html>
<html>
  <head>
    <style type="text/css">
      .titleText {
        color: green;
        font-family:arial;
      }
    </style>
    <script>
    </script>
    <script>
      $(function(){
        $("<h1>Have A Nice Day</h1>")
          .appendTo("body")
          .attr("class", "titleText");
      });
    </script>
  </head>
  <body>
  </body>
</html>
```

代码片段 *generating-html.txt*

4.5 小结

经过本章的学习，你应该已经非常清楚地认识到 jQuery 选择器是多么强大和高效。在选取元素或操作 DOM 时，协调、高效地使用各种类型的 jQuery 选择器，可以极大地降低代码的复杂性。

现在，我们已经回顾了 JavaScript 语言的核心技术、学习了 jQuery 包装器、学习了如何编写简洁的选择器表达式、以及修改元素内容的各种办法，我们对 jQuery 的知识已经有了一个坚实的基础。如果将这些概念应用于实际的开发工作，那么你的 JavaScript 代码将非常简洁，并且具有良好的可读性。



第 5 章

事件处理

本章内容

- 理解浏览器的事件模型
- 理解 jQuery 如何处理事件
- 使用 jQuery 的事件

任何基于 GUI 的现代应用程序本质上都是事件驱动的，Web 应用程序也不例外。所有事件驱动的应用程序都采用相同的工作模式：建立事件处理机制、等待相关事件发生(比如鼠标单击)、对该事件作出响应。除了普通的鼠标单击和键盘击键事件之外，还有页面加载事件和其他一些语义化的事件，比如最新添加的滑动手势(swipe gestures)事件，由于 iPhone 和 Android 等触屏设备的流行，手势事件已经变得非常重要。事件是 Web 应用程序的基础，如果不使用事件，那么 JavaScript 的使用将受到严重制约。在 jQuery 的文档中，把事件方法分类为文档加载事件、表单事件、事件处理程序绑定、鼠标事件、浏览器事件、与 Event 对象有关的方法和键盘事件。

在一个开发人员的职业生涯中，几乎不可能没有听说过各种跨浏览器的问题，这些问题困扰着 Web 开发人员。不同浏览器如何管理事件，就是最令人头疼的跨浏览器问题之一。

本章将回顾基本的 JavaScript 事件处理模型，包括 DOM level 0 和 level 2、事件传播、事件冒泡以及 Internet Explorer 的事件模型，以便你更好地理解事件背后的工作原理。随后将把这些知识转化为 jQuery 的事件处理机制，介绍 jQuery 如何规范化所有这些浏览器的差异。最后还将把这些知识应用于一个示例应用程序。

5.1 理解浏览器的事件模型

在学习 jQuery 如何处理事件并消除跨浏览器兼容性的问题之前，我们先回顾一下

JavaScript 的事件模型。JavaScript 最早是在 1995 年引入的。如果钻入 DeLorean(电影《回到未来》中的时光穿梭机)并回到 1996 年,那时 Netscape Navigator 和 Microsoft Internet Explorer 3 在年轻人中正风靡一时。表 5-1 列出了 Web 编程早期使用的不同事件模型。

表 5-1 DOM 级别事件

DOM 级别	描 述
0	由 Netscape 开发的最早的事件模型。所有浏览器都支持该事件模型
2	标准事件模型,比 level 0 更高级,但 IE 浏览器不支持该级别的事件模型
IE Model	IE 浏览器使用的一个独立的事件模型

DOM level 0 指的是 Netscape Navigator 2.0 所支持的文档模型,随后的很多浏览器都采用了这一模型。虽然该模型不是一个正式的标准,但所有主流浏览器都与 level 0 兼容。DOM 提供了一个访问和修改 HTML 文档内容、样式和结构的接口。DOM level 0 中的事件有两种处理方式,一是内联方式(inline),即在元素的属性中添加一个以 on 开头的事件属性,并将 JavaScript 代码作为该属性的值;二是在 JavaScript 代码中选中该元素,然后在该元素上绑定一个匿名函数作为事件处理程序。下面的代码示例演示了第一种事件处理方法:



可从
Wrox.com
下载源代码

```
<html>
<head>
  <style type="text/css">
    .clickDiv {
      width:100px;
      height:100px;
      background-color:blue;
    }
  </style>
  <script type="text/javascript">
    function hitMe(){
      var txtNode = document.createTextNode(" clicked ");
      var br = document.createElement("br");
      document.getElementsByTagName("body")[0].appendChild(txtNode);
      document.getElementsByTagName("body")[0].appendChild(br);
    }
  </script>
</head>
<body>
  <div class="clickDiv" onclick="hitMe();">
</div>
</body>
</html>
```

代码片段 onclick.txt

显然,上面例子中的 JavaScript 是侵扰式的。它在 onclick 事件上绑定了 hitMe()方法。

还可以将上面这个例子改写为下面的形式：



```
<html>
  <head>
    <style type="text/css">
      .clickDiv {
        width:100px;
        height:100px;
        background-color:blue;
      }
    </style>
    <script type="text/javascript">
      document.getElementById("clickDiv").onclick = function(event){
        document.getElementsByTagName("body")[0].appendText("clicked")
      };
    </script>
  </head>
  <body>
    <div id="clickDiv" class="clickBox">Click Here</div>
  </body>
</html>
```

代码片段 better-onclick.txt

改写后的代码将控制逻辑从文档结构中分离出来。这样的代码依然还会遇到一些严重的跨浏览器问题。绑定到 onclick 事件上的函数接收一个 event 对象作为参数。虽然绝大多数浏览器接收标准的 event 对象作为参数，但 Internet Explorer 使用的是 window.event，即在 Internet Explorer 中将 event 对象绑定到了全局的 window 对象。使用特性检测要比浏览器检测更好，因此添加了特性检测后的事件处理代码如下所示：

```
var evt = (window.event ? window.event : event);
```

另外，标准 event 对象与 IE 的 Event 对象是不同的，它们支持的属性集略有不同。对于跨浏览器脚本，这令人非常烦恼并打破了浏览器之间的兼容性。表 5-2 列出了 IE 的 Event 对象与标准 Event 对象之间的相同点或不同点。

表 5-2 Firefox 浏览器 Event 对象的兼容性

名 称	属性或方法	IE 对应的属性或方法
stopPropagation()	方法	cancelBubble()
layerX、layerY	属性	offsetX、offsetY
Target	属性	srcElement
Type	属性	Type
preventDefault()	方法	returnValue()

(续表)

名 称	属性或方法	IE 对应的属性或方法
altKey、ctrlKey、shiftKey	属性	altKey、ctrlKey、shiftKey
Button	属性	Button
charCode	属性	keyCode
relatedTarget	属性	fromElement、toElement
Bubbles	属性	N/A

有时,我们想在同一个元素的 onclick 事件上绑定多个不同的事件处理程序。在 DOM level 0 中,后绑定的事件处理程序将覆盖之前绑定在事件上的事件处理程序。

在 DOM level 2 的事件中为此提供了解决方案。你可能已经注意到,没有 DOM level 1 级别的事件,这是因为不存在 DOM level 1 这样的标准。另一方面,DOM level 2 是一个比 DOM level 0 更高级的模型,除了 IE 8 和更低版本的 IE 浏览器之外,其他绝大多数浏览器都支持 DOM level 2 模型。稍后还将介绍更多 IE 浏览器与 DOM level 2 的差异。在 DOM level 2 中,不再采用把事件处理程序赋值给对象属性的方式来绑定事件,DOM 中的每一个页面对象都具有一个名为 `addEventListener()` 的方法,它允许开发人员在一个 DOM 元素上添加多个事件处理程序。例如:



```
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script>
    $(function() {
      var eventType = "click";
      var useCapturePhase = false;
      var handler1 = function(event) { /* handler code here */};
      var handler2 = function(event) { /* handler code here */};

      $("#element").addEventListener(eventType,
        handler1, useCapturePhase);
      $("#element").addEventListener(eventType,
        handler2, useCapturePhase);
    });
  </script>
</head>
<body>
</body>
</html>
```

代码片段 `addEventListener.txt`

`addEventListener()` 方法接收三个参数: 第一个参数是事件类型(在本例中即 “click”),

第二个参数是用于处理该事件的函数，第三个参数是一个布尔值，用于事件捕获阶段。什么是“事件捕获阶段”呢？你可能会问。

5.1.1 事件的捕获或冒泡

请考虑这样的情形：具有相同事件类型的元素嵌套在一起，比如都具有 `onclick` 事件的元素。应该首先执行哪一个元素的 `onclick` 事件处理程序呢？内部的元素先执行，还是外部元素先执行？最初，Netscape 认为事件处理应该由外向内依次进行，这一过程称为事件捕获。如图 5-1 所示。

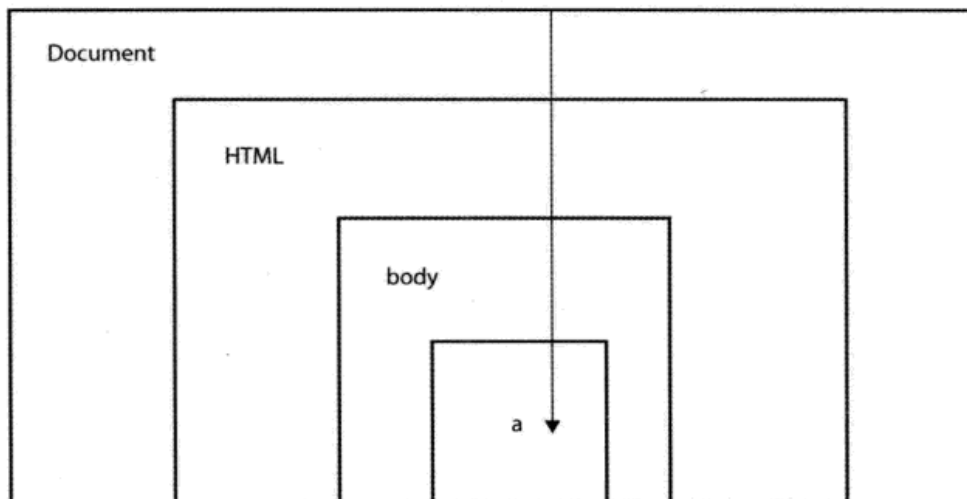


图 5-1

Microsoft 则采用了相反的事件处理顺序，即先执行内部元素的 `onclick` 事件处理程序，然后再执行外部元素的 `onclick` 事件处理程序，这一过程称为事件冒泡阶段，如图 5-2 所示。

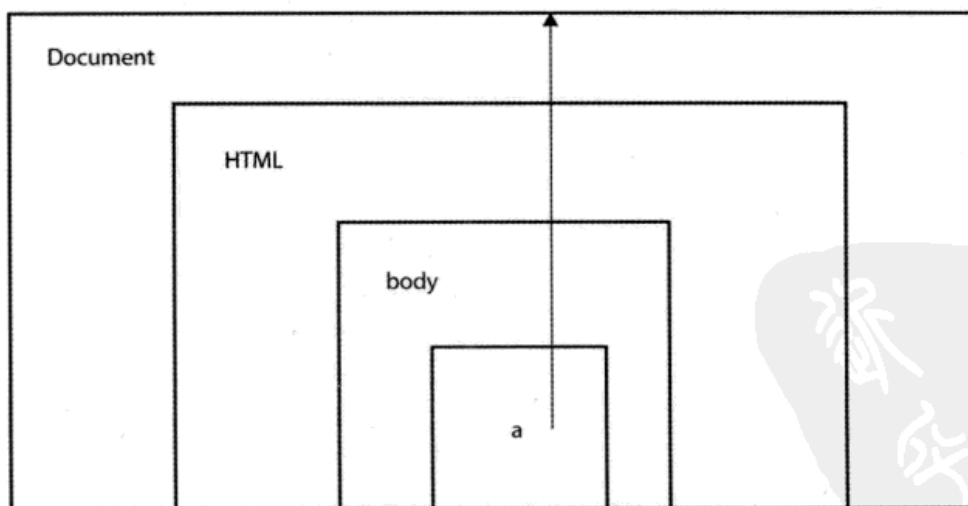


图 5-2

传入 `addEventListener()` 方法的第三个参数是一个布尔值，它是一个标记，用于决定是否使用事件捕获阶段。

但是，开发人员可能根本并不想让事件传播或冒泡发生，这可以在 JavaScript 中进行

控制。对于那些遵循 W3C 标准的浏览器。只需要调用一个名为`.stopPropagation()`的方法即可。对于较低版本的 IE 浏览器，可以将 `event` 对象的 `cancelBubble` 属性设置为 `true`，以阻止事件向上冒泡。

下面的例子演示了事件捕获：



```
<html>
<head>
  <style type="text/css">
    .div1Class {
      width: 300;
      height: 300;
      background-color: blue;
    }

    .div2Class {
      width: 200;
      height: 200;
      background-color: green;
    }

    .div3Class {
      width: 100;
      height: 100;
      background-color: red;
    }
  </style>

  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script>
    $(function(){
      function f(id){
        document.body.innerHTML += "<p>executing handler for div " + id
+ "</p>";
      }

      document.getElementById("div1").onclick = function(){
        f("1");
      }

      document.getElementById("div2").onclick = function(){
        f("2");
      }

      document.getElementById("div3").onclick = function(){
        f("3");
      }
    });
  </script>
```

```

</head>
<body>
  <div id="div1" class="div1Class">
    <div id="div2" class="div2Class">
      <div id="div3" class="div3Class">
      </div>
    </div>
  </div>
</body>
</html>

```

代码片段 *stop-propagation.txt*

在 Google Chrome 浏览器中测试上面的例子，可以看到图 5-3 所示的结果。在点击了 div 3 之后，每一个 div 元素的 onclick 事件处理程序都被依次调用。

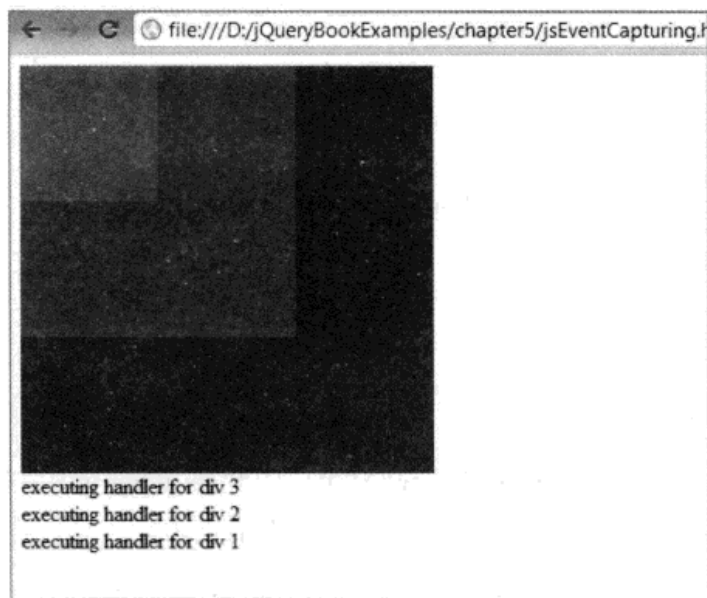


图 5-3

IE 8 以及更低版本的 IE 浏览器并不支持 DOM level 2，这与大多数其他浏览器不同。例如，IE 浏览器中事件模型使用的是 `attachEvent` 方法，而不是 `addEventListener` 方法。前面已经提到过，与 W3C 标准兼容的其他浏览器相比，IE 的 Event 对象并未提供与之完全相同的方法和属性，甚至事件的名称也不同。IE 提供了一种与事件冒泡类似的事件传播机制，但与事件冒泡又并非完全相同。

5.2 理解 jQuery 中的事件处理机制

一旦谈到事件传播、事件模型的不统一等浏览器的差异问题，就足以使任何开发人员抓狂。顺理成章地，接下来的发展就是抽取出不同事件处理机制存在的差异，这正是 jQuery 要实现的功能。在 jQuery 中可以使用不同的方式绑定事件处理程序：`.bind()` 方法、`.live()`

方法, 使用一些事件方法, 比如.click()、.load(), 或者.mouseout()方法等, 以及 jQuery1.7 中新引入的 on()和 off()方法。

与 DOM level 2 类似, 使用.bind()方法可以在同一个事件上绑定多个事件处理程序:



```
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script>
    $(function(){
      $("#aDiv").bind('click', function(){
        console.log("Handler 1");
      });

      $("#aDiv").bind('click', function(){
        console.log("Handler 2");
      });
    });
  </script>
</head>
<body>
  <div id="aDiv" class="boxDiv">Press Me
</div>
</body>
</html>
```

代码片段 bind.txt

jQuery 从事件名称中移除了多余的"on"前缀, 因此在 jQuery 中应该使用 click()取代 onclick()。在上面的例子中, 当在 ID 为 aDiv 的元素上点击时, 可以在控制台中看到输出了两条信息。由于 jQuery 在后台处理了所有的细节问题, 因此.bind()、.click()等 jQuery 方法都与使用哪一种浏览器来解释代码无关。这让人如释重负!

jQuery 的事件处理机制对所有不同类型的事件进行了统一的命名, 而且独立于浏览器。与 DOM level 2 模型类似, jQuery 允许在同一个元素和事件类型上绑定多个事件处理程序。在 jQuery 中, 传入事件处理程序的 Event 对象也是规范化的, 它的类型是 jQuery.Event, jQuery 确保 Event 对象的属性独立于浏览器或平台, 这些属性如表 5-3 所示:

表 5-3 Event 对象独立于浏览器或平台的属性

altKey	Detail	pageY
attrChange	eventPhase	prevValue
attrName	fromElement	relatedNode
Bubbles	Handler	relatedTarget
Button	keyCode	Screen
Cancelable	layerX	shiftKey

(续表)

charCode	layerY	srcElement
Client	metaKey	Target
Client	newValue	toElement
ctrlKey	Offset	View
currentTarget	originalTarget	wheelDelta
Data	pageX	which

5.3 使用 jQuery 进行事件处理

jQuery 通过 `.bind()` 方法将一个函数绑定到一个事件, `.bind()` 方法接收 3 个参数: 第一个参数是事件类型、第二个参数是数据、第三个参数是一个函数处理程序。例如, 要绑定一个 `mouseover` 事件, 可以采用下面的 `.bind()` 方法:

```
$("#objId").bind('mouseover', function(event){
    //要执行的代码放在这里...
});
```

在上面的代码中, 首先通过 jQuery 选择了 id 为 `objId` 的元素, 然后在 `mouseover` 事件上绑定了一个匿名函数。

表 5-4 列出了所有可用的 jQuery 事件处理程序。

表 5-4 jQuery 事件

<code>.blur()</code>	<code>.mouseenter()</code>
<code>.focus()</code>	<code>.mousemove()</code>
<code>.select()</code>	<code>.mouseout()</code>
<code>.submit()</code>	<code>.mouseover()</code>
<code>.click()</code>	<code>.mouseup()</code>
<code>.dblclick()</code>	<code>.toggle()</code>
<code>.focusin()</code>	<code>.error()</code>
<code>.focusout()</code>	<code>.resize()</code>
<code>.hover()</code>	<code>.scroll()</code>
<code>.mousedown()</code>	

另外, 可以使用 `.unbind()` 方法移除任何事件处理程序的绑定, `.unbind()` 方法接收一个字符串作为参数, 该字符串就是要解除绑定的事件类型名称。比如对于上面的例子, 可以使用下面的代码移除绑定:

```
$("#selectorHere").unbind("mouseover");
```

这将从选中的元素上移除所有 `mouseover` 类型的事件。如果想移除某一类特定的事件绑定，也可以为事件定义名称空间。

由于 JavaScript 或 jQuery 可以在运行时动态插入 DOM 元素，因此在编写代码时，想要绑定事件处理程序的元素可能还不存在于页面的 DOM 中。如果发生这种情况，`.bind()` 方法将不再有效，而应该使用对应的 `.live()` 方法，无论元素是否已经存在，该方法都可以为其绑定一个事件处理程序。对于在运行时插入的 DOM 元素，`.live()` 方法非常有用。比如下面的例子，它使用 `.live()` 方法为页面中还不存在的锚元素绑定了一个 `click` 事件处理程序。当 `.ready()` 方法执行时，锚元素将动态地添加到页面中，并绑定到 `click` 事件处理程序。



可从
Wrox.com
下载源代码

```
<html>
<head>
  <scriptsrc="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script>
    $(function(){
      //虽然元素现在还不存在，但可以创建事件处理程序
      $("#anchor").live("click", function(event){
        console.log("I have a handler");
      });

      $(document).append("<a id='anchor'> I go no where </a>")
    });
  </script>
</head>
<body>
</body>
</html>
```

代码片段 live.txt

与 `.unbind()` 方法类似，`.live()` 方法也具有一个与之对应的 `.die()` 方法，用于移除由 `.live()` 方法设置的事件处理程序。与其他 jQuery 方法不同，`.live()` 方法无法链接调用。在想继续使用链式调用的场合，请使用 `.delegate()` 方法代替 `.live()` 方法，它同样可以为还不存在的元素绑定一个事件处理程序。



可从
Wrox.com
下载源代码

```
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script>
    $(function(){
      $("body").delegate("p", "click", function(){
        console.log('ouch');
      }).css("color", "green");
    });
  </script>
```



```

</head>
<body>
<p>Hit Me!</p>
</body>
</html>

```

代码片段 *delegate.txt*

与`.unbind()`和`.die()`方法的关系类似，`.delegate()`方法也具有一个对应的`.undelegate()`方法。例如，要移除前面例子中绑定的事件处理程序，可以使用下面的代码：

```
$("#body").undelegate("p","click");
```

在某些情况下，我们可能希望一个事件处理程序只执行一次，或者最多只执行一次。可以使用`.one()`方法来实现这样的功能，下面的例子演示了`.one()`方法的使用。对于第一个

元素，`click`事件处理程序只执行一次。



```

<html>
<head>
<style type="text/css">
    .div1 {
        width : 100;
        height : 100;
        background-color: blue;
    }

    .div2 {
        width : 100;
        height : 100;
        background-color: red;
    }
</style>
<script src="http://code.jquery.com/jquery-1.7.1.js"></script>
<script>
    $(function(){
        $(".div1").one("click", function(){
            $("#body").append("<p>clicked div 1 </p>");
        });

        $(".div2").bind("click", function(){
            $("#body").append("<p>clicked div 2 </p>");
        });
    });
</script>
</head>
<body>
<div class="div1"></div>
<div class="div2"></div>

```

```

</body>
</html>

```

代码片段 one.txt

图 5-4 显示了该示例的运行结果。

jQuery 还提供了创建事件处理程序的快捷方法。比如下面的代码：

```

$("#el").bind('click', function(event){
    //执行某些工作
});

```

可以重写为：

```

$("#el").click(function(event){
    //执行某些工作
});

```

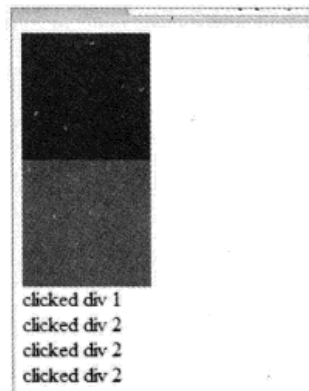


图 5-4

对于 hover、mouseout、mouseover 等事件也是如此。在下面的例子中，通过链接两个快捷事件方法 mouseover 和 mouseout，创建了一个经典的“鼠标感应”效果：



```

<html>
<head>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script>
    $(function(){
      $("#text").mouseover(function(){
        $(this).css("text-decoration","underline");
      }).mouseout(function(){
        $(this).css("text-decoration","none");
      });
    });
  </script>
</head>
<body>
  <p id="text">I go no where</p>
</body>
</html>

```

代码片段 mouseover.txt

我们还可以用编程方式或手工方式执行已经存在的事件处理程序。也就是说，不必事件实际地发生，就可以用编程方式触发一个事件处理程序，对于测试这非常有用。在下面的例子中，三个 div 元素(为每一个 div 设置了一个不同的背景色以便于区分)分别绑定了一个 mouseover 事件处理程序，在事件处理程序中，为嵌套在 div 元素中的<p>元素创建了淡入和淡出效果。在三个 div 元素下面，是一个命令按钮，当点击该命令按钮时将触发.trigger()

方法的执行，`.trigger()`方法的效果是使所有 `mouseover` 事件处理程序一起执行。`.trigger()`方法是在一个包装集上调用的，它接收两个参数：第一个参数是一个字符串，它表示要触发的事件类型，在本例中即 `mouseover`；第二个参数是一个要传入的附加参数的数组，也可以传入一个 `jQuery.Event` 对象。默认情况下，通过调用 `.trigger()` 方法触发执行的事件，将会向上冒泡。



可从
Wrox.com
下载源代码

```
<html>
<head>
  <style type="text/css">
    div {
      padding : 10 10 10 10;
      width : 100;
      height : 100;
    }
    .div1 {
      background-color : blue;
    }
    .div2 {
      background-color : yellow;
    }
    .div3 {
      background-color : green;
    }
  </style>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script>
    $(document).ready(function() {
      $("p").hide();

      $(".div1", this).mouseover(function() {
        $(this).find("p").fadeIn().fadeOut();
      });

      $(".div2", this).mouseover(function() {
        $(this).find("p").fadeIn().fadeOut();
      });

      $(".div3", this).mouseover(function() {
        $(this).find("p").fadeIn().fadeOut();
      });

      $("input").click(function() {
        $("div").trigger("mouseover");
      });
    });
  </script>
</head>
<body>
```



```
<div class="div1">
  <p>Here</p>
</div>
<div class="div2">
  <p>Here</p>
</div>
<div class="div3">
  <p>Here</p>
</div>
<input type="button" value="run with trigger"></input>
</body>
</html>
```

代码片段 trigger.txt

图 5-5 显示了前面例子所创建的效果。

.trigger() 方法调用的是浏览器事件。例如，执行 `$("#myButton").trigger("click");` 将调用 JavaScript 默认的 `onclick` 方法。此外还可以使用 `.triggerHandler()` 方法，顾名思义该方法执行附加的事件处理程序，而不是原生的事件处理方法。`.triggerHandler()` 方法接收的参数与 `.trigger()` 方法相同。但与 `.trigger()` 方法不同的是，`.triggerHandler()` 方法只执行第一个匹配对象的事件处理程序，而不是整个匹配集中每一个元素的事件处理程序。另外，`.triggerHandler()` 方法触发的事件，也不会向上冒泡。

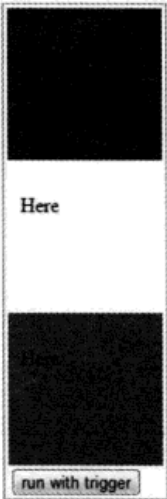


图 5-5

表 5-5 汇总了 jQuery 的附加事件处理程序。

表 5-5 附加的事件处理程序

方 法	描 述
.bind()	绑定一个事件处理程序
.delegate()	为当前已存在或运行时才存在的元素绑定一个事件处理程序
.die()	移除使用.live()方法创建的事件处理程序
.live()	为当前已存在或运行时才存在的元素绑定一个事件处理程序
.one()	附加一个最多只执行一次的事件处理程序
.proxy()	接收一个函数并返回另外一个属于特定上下文的新函数
.trigger()	为某一事件类型执行附加到元素上的事件处理程序
.triggerHandler()	为特定事件执行所有的事件处理程序
.unbind()	移除使用.bind()创建的事件处理程序
.undelegate()	移除使用.delegate()方法创建的事件处理程序

5.4 使用事件

在下面的例子中，不仅使用了关于事件的知识，而且还融合了到目前为止我们所积累的 jQuery 知识。对于任何一个数据库驱动型的 Web 应用程序来说，最常见的特性之一就是具有一个注册表单。在注册表单中通常需要输入电子邮件、社交网络信息或任何你想接收信息的地址，至少也要求你提供一个电子邮件和一个密码。在下面这个应用程序示例中，将创建一个注册表单的骨架，可以很容易地用它来实现更加复杂、完备的用户注册表单。该注册表单要求输入用户名、电子邮件、密码和确认密码。尽管该应用程序并未实际保存这些数据，但它提供了一个命令按钮用于对输入进行验证。所有字段都要求输入数据，并且为了检验密码的输入，密码必须与确认密码完全相同。此外，用户还可以添加地址信息，在本例中用简单的下拉列表框供用户选择美国的州和相应的城市信息。

单击 Add Location 命令按钮将生成一个 state 下拉列表框和一个 city 下拉列表框，二者是关联在一起的——当选中的州发生改变时，可供选择的的城市列表也随之改变。当然，可供选择的州和城市列表一点也不复杂，甚至不太实用，因此例子中仅仅考虑了美国的几个州和这几个州中的几个城市。但对于演示该示例的功能已经足够了。图 5-6 显示了完成后的注册表单。

图 5-6

首先，从编写基本的标记代码开始。这些代码都已经很老套了。在开头一些公式化的规范 HTML 之后，<body>元素包含了一个用于显示任何错误信息的 div 元素、一个带有文本输入框和两个密码输入框的表单。在底部是两个命令按钮，请注意，<body>元素中仅包含 HTML 元素，没有任何 JavaScript 代码。这些 HTML 代码如下所示：

```
<!DOCTYPE html>
<html>
  <head>
    <title>User Registration and Validation</title>
    <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
    <script>
      $(function() {
        //jQuery 代码放在这里
      });
    </script>
  </head>
```

```

<body>
  <div id="msg"></div>
  <form name="userRegistrationForm">
    <label for="userName">User</label><input type="text"
name="userName" ><br/>
    <label for="email">Email</label><input type="text" name="email"
/><br/>
    <label for="password">Password</label><input type="text"
name="password"
    /><br/>
    <label for="chkPassword">Re-enter Password</label>
    <input type="text" name="chkPassword" /><br/>
    <input type="button" name="validate" value="Validate Inputs" />
  </form>
  <input type="button" name="addLocation" value="Add Location" />
</body>
</html>

```

`$(function(){...});`等价于`.ready()`方法，在页面的所有 DOM 元素加载完成之后，它将执行函数体中所包含的代码，不必等待文档中所有附属文件都全部下载完成。在 `$(function(){...});` 的函数体中，首先添加的行为就是检验各个输入字段。在 `validate` 命令按钮的 `click` 事件触发后，将执行检验工作。使用 jQuery 的属性选择器选中 `validate` 按钮，使之成为一个包装对象，然后绑定 `click` 事件、将一个匿名函数作为 `click` 事件的处理程序，无论何时，只要点击了 `validate` 按钮，就对 `id` 为 `msg` 的 `div` 元素使用 `.html()` 方法移除旧有消息。之后，使用 `.val()` 方法分别抽取各个输入框的值。

使用 `&&` 操作符可以简化条件判断，当其中有任何一个值为空(`null`)时，就在运行时将一些提示文本追加到 `id` 为 `msg` 的 `div` 元素中，并将字体颜色的 CSS 样式修改为红色。第二个检查步骤，就是使用等同操作符(`===`)确认两次输入的密码是否完全相同。与之前的验证一样，如果两次输入的密码不一致，则将相应的提示消息追加到 `id` 为 `msg` 的 `div` 元素中。这两条提示消息每次只会显示其中一条，不会同时将两条提示消息都显示出来。图 5-7 和图 5-8 分别显示出现这两条提示消息的页面。

```

...
$( 'input[name="validate"]' ).click(function(){
    //清除 msg 中的信息
    $("#msg").html("");

    //获取各个输入框的值
    var userName = $( 'input[name="userName"]' ).val();
    var email = $( 'input[name="email"]' ).val();

    var pass1 = $( 'input[name="password"]' ).val();
    var pass2 = $( 'input[name="chkPassword"]' ).val();

    //不允许出现空值
    var hasValue = userName && email && pass1 && pass2;

```



```

    if( !hasValue ){
        $("#msg")
            .append("All Fields are required.")
            .css("color","red");
        return false;
    }

    //检查两次输入的密码是否完全相同
    var passwordMatch = false;
    if( pass1 === pass2 ) {
        passwordMatch = true;
    }

    if( !passwordMatch ){
        $("#msg").append("<p>Passwords don't match.</p>").css("color",
"red");
        return false;
    }
}
});
...

```

图 5-7

图 5-8

接下来就是最有技巧的部分：页面中不存在任何一个下拉列表框，直到用户单击了 **Add Location** 命令按钮。因此，下拉列表框元素必须由 jQuery 在运行时生成，这还意味着必须使用 `.live()` 方法来创建事件处理程序，而不能使用 `.bind()` 方法。

```

$("#input[name='addLocation']").click(function(){
    $("body" )
        .append("<select name='stateCombo'>"
            + "<option>Select State</option></select>");
});

```

对于上面这段代码，在 **Add Location** 命令按钮的 `click` 事件处理程序(监听器)中，使用了 `.append()` 方法来创建 `<select>` 元素。如果多次单击该命令按钮将会连续地在文档的 `<body>` 元素中追加重复的 `<select>` 元素。为了避免出现这一情况，可以在第一次单击 **Add Location** 命令按钮后禁用该按钮。相应代码如下所示：

```

$(this).attr("disabled", "disabled");

```

在本示例中仅提供几个有限的地址选项，我们只选取了三个州作为例子：California、

Florida 和 New York。将表示这些州名的字符串保存在一个数组中，当生成<select>元素中的<option>选项列表时，将遍历该数组以生成相应的 HTML 代码。同时，将第二个<select>元素(用于选择城市)添加到文档的<body>元素中，但目前它不包含任何数据。

```
//添加几个州名作为例子
var states = ["California", "Florida", "New York"];

$.each(states, function(index, value){
    $("[name='stateCombo']")
        .append("<option value='"+index+"'> " + value + "</option>");
});

//添加一个空的<select>元素，用于选择城市
$("body")
    .append("<select name='cityCombo'><option>Select
        City</option></select>");
});
```

请注意，为了生成 HTML 字符串，必须使用一些字符串的连接操作。在这里最好使用可读性更好的 jQuery 字符串模版，这将在第 11 章中进行介绍。在接下来的代码片段中，使用了.live()方法，将“change”事件绑定到表示州名列表的<select>元素，或者说下拉列表框。采用.live()方法绑定事件，就不必担心选中的元素在加载时是否已经实际存在。

对于三个不同的州，定义了三个字符串数组分别用于保存各个州对应的城市名称，这三个数组分别保存在三个临时变量中。根据从州名称下拉列表框中选中了哪一个州(0 代表 California, 1 代表 Florida, 2 代表 New York)，决定使用哪一个包含城市名列表的数组，并使用该数组来生成代表城市下拉列表框的<select>元素。每次当选中的州发生改变时，代表城市的<select>元素，即 cityCombo 元素将被清空，换句话说，将从 DOM 中移除该<select>元素的所有子元素。

```
//使用.live()方法，因为州下拉列表框在页面加载时还不存在
$("[name='stateCombo']").live("change", function(event){
    //获取选中的州名称，并定义城市名称数组
    var selectedState = $(this).val();

    var CA_Cities = ["San Francisco", "Los Angeles", "Mountain View"];
    var FL_Cities = ["Fort Lauderdale", "Miami", "Orlando"];
    var NY_Cities = ["New York", "Buffalo", "Ithica"];
    var cities = [];

    if(selectedState == 0){
        cities = $.extend([], CA_Cities);
    } else if(selectedState == 1){
        cities = $.extend([], FL_Cities);
    } else if(selectedState == 2){
        cities = $.extend([], NY_Cities);
    }
});
```

```

//清除 cityCombo 中任何之前的值
$("#[name='cityCombo']").empty();
$.each(cities, function(index, value){
    $("#[name='cityCombo']").append("<option value='"+index+"'>"
                                     +value+"</option>");
});
});

```

现在, 已经可以将所有这些代码片段合成到完整的应用程序中。下面的代码块就是该例子的完整代码。其中还有一些地方需要改进, 比如电子邮件输入框的掩码, 也许可以使用第三方插件进行改进。但对于演示如何为动态生成的元素创建事件处理程序以及如何使用一些更常见的事件——比如鼠标单击事件和 `ready` 事件——而言, 这个示例已经绰绰有余。



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
<head>
<title>User Registration and Validation</title>
<script src="http://code.jquery.com/jquery-1.7.1.js"></script>
<script>
$(function(){
    $('#input[name="validate"]').click(function(){
        //清除 msg 中的信息
        $("#msg").html("");

        //获取各个输入框的值
        var userName = $('#input[name="userName"]').val();
        var email = $('#input[name="email"]').val();
        var pass1 = $('#input[name="password"]').val();
        var pass2 = $('#input[name="chkPassword"]').val();

        //不允许出现空值
        var hasValue = userName && email && pass1 && pass2;
        if( !hasValue ){
            $("#msg")
                .append("All Fields are required.")
                .css("color","red");
            return false;
        }

        //检查两次输入的密码是否完全相同
        var passwordMatch = false;
        if( pass1 === pass2 ){
            passwordMatch = true;
        }

        if( !passwordMatch ){
            $("#msg").append("<p>Passwords don't match.</p>").css("color",

```



```

    "red");
    return false;
  }
});

$( "input[name='addLocation']" ).click(function(){
  $("body" ).append( "<select name='stateCombo'><option>"
    + "Select State</option></select>" );

  //禁用 Add Location 按钮，避免重复点击该按钮添加重复的下拉列表框
  $(this).attr("disabled", "disabled");

  //添加几个州名作为例子
  var states = ["California", "Florida", "New York"];
  $.each(states, function(index, value){
    $("[name='stateCombo']").append("<option value='"
      + index
      + "'>"
      + value
      + "</option>");
  });

  //添加一个空的<select>元素，用于选择城市
  $("body").append("<select name='cityCombo'>"
    + "<option>Select City</option></select>");
});

//使用.live()方法，因为州下拉列表框在页面加载时还不存在
$("[name='stateCombo']").live("change", function(event){
  //获取选中的州名称，并定义城市名称数组
  var selectedState = $(this).val();

  var CA_Cities = ["San Francisco", "Los Angeles", "Mountain View"];
  var FL_Cities = ["Fort Lauderdale", "Miami", "Orlando"];
  var NY_Cities = ["New York", "Buffalo", "Ithica"];
  var cities = [];

  if(selectedState == 0){
    cities = $.extend([], CA_Cities);
  } else if(selectedState == 1){
    cities = $.extend([], FL_Cities);
  } else if(selectedState == 2){
    cities = $.extend([], NY_Cities);
  }

  //清除 cityCombo 中任何之前的值
  $("[name='cityCombo']").empty();
  $.each(cities, function(index, value){
    $("[name='cityCombo']").append("<option value='"
      + index

```

```

+ "'>"
+ value
+ "</option>");

});
});
});
</script>
</head>
<body>
<div id="msg"></div>
<form name="userRegistrationForm">
  <label for="userName">User</label>
  <input type="text" name="userName" /><br/>
  <label for="email">Email</label>
  <input type="text" name="email" /><br/>
  <label for="password">Password</label>
  <input type="password" name="password" /><br/>
  <label for="chkPassword">Re-enter Password</label>
  <input type="text" name="chkPassword" /><br/>
  <input type="button" name="validate" value="Validate Inputs" />
</form>
<input type="button" name="addLocation" value="Add Location" />
</body>
</html>

```

代码片段 event-handling.txt

5.5 jQuery 新的事件 API

自从 jQuery 1.7 版本开始, jQuery 已经努力地简化了用于设置事件的 API。尽管我们依然可以使用原来的方法,比如 .bind() 和 .unbind()、.live() 和 .die(), 以及设置 .click() 和 .blur(), 但 jQuery 已经引入了一个统一的 API, 它可以用一个统一的 API 来绑定所有这些不同的功能。jQuery 引入的两个新方法是 .on() 和 .off() 方法。这两个新方法代表了更好的 API, 因此值得学习它们的使用方法, 并在启动任何新项目时, 直接使用推荐的新 API。

.on() 方法的美妙之处, 在于它用一个单一的、灵活的 API, 取代了本章中介绍的所有事件方法的功能。下面的代码示例演示了 .on() 方法的基本使用, 它使用 .on() 方法为 DOM 中已经存在的一个元素绑定了一个 click 事件。在本例中, .on() 方法接收两个参数, 第一个参数是一个事件类型(eventType), 第二个参数是要执行的事件处理程序(handler)。

为了与直接使用 .bind("click") 或 .click() 方法的代码作比较, 下面的例子改写了本章前面介绍的 .bind() 方法的示例。



可从
Wrox.com
下载源代码

```

<html>
<head>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>

```

```

<script>
  $(function() {
    $("#aDiv").on('click', function() {
      console.log("Handler 1");
    });

    $("#aDiv").on('click', function() {
      console.log("Handler 2");
    });
  });
</script>
</head>
<body>
  <div id="aDiv" class="boxDiv">Press Me</div>
</body>
</html>

```

代码片段 on-click.txt

除了基本的 `on()` 方法示例之外，它还提供了与 `delegate()` 和 `live()` 方法相同的功能，并且统一使用一种语法来实现这些功能。要使用 `on()` 方法实现像 `delegate()` 方法或 `live()` 方法一样的功能，只需要在第 2 个参数位置，添加一个可选的选择器作为参数，该选择器代表了要绑定事件的目标元素。在下面的第一个例子中，使用 `on()` 方法取代了本章前面例子中介绍的 `live()` 方法。在该示例中，`on()` 方法用于监听文档的 `click` 事件，如果 `click` 事件源自于一个 ID 为 `#anchor` 的元素，则触发指定事件处理程序的执行。由于将监听器设置在文档上，因此无论 `#anchor` 元素是否出现，它都能正常工作。



可从
Wrox.com
下载源代码

```

<html>
<head>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script>
    $(function() {
      //虽然元素现在还不存在，但可以创建事件处理程序
      $(document).on("click", "#anchor", function(event) {
        console.log("I have a handler");
      });
      $("body").append("<a id='anchor'> I go no where </a>");
    });
  </script>
</head>
<body>
</body>
</html>

```

代码片段 on-live.txt

可以使用 `on()` 方法取代 `delegate()` 方法，它所使用的语法与前面例子中取代 `live()` 方法的语法完全相同。这是新的、简化的事件 API 所带来的好处之一。开发人员不必再分别记忆两个独立的方法和相关的参数，只需要记住一个 `on()` 方法并调整相应的参数即可。如果确实需要将事件绑定于整个页面中的某一种元素，只需要在 `document` 对象上应用 `on()` 方法，并传入元素的标记名(比如 `p`)或一个选择器即可，如果想遵循性能更好的 `delegate` 模式，只需要简单地将 `on()` 应用于一个选择范围更精确的选择器即可。

可以使用 `on()` 方法取代之前的例子中的 `delegate()` 方法，重新编写该示例，相应代码如下所示。在下面的例子中，`on()` 方法用于监听 `id` 为 `#delegate` 的 `div` 元素上的 `click` 事件，如果点击了 `<p>` 元素，则触发指定的事件处理程序。



```
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script>
    $(function() {
      $("#delegate").on("click", "p", function() {
        console.log('ouch');
      }).css("color", "green");
    });
  </script>
</head>
<body>
  <div id="delegate">
    <p>Hit Me!</p>
  </div>
</body>
</html>
```

代码片段 `on-delegate.txt`

移除事件是绑定事件的逆向过程，要移除事件非常简单，只需要使用 `off()` 方法即可。它的使用方法与 `unbind()` 或 `die()` 方法类似。下面的代码说明了如何移除前面示例中绑定的事件：

```
$("#delegate").off("click", "p");
$(document).off("click", "#anchor");
$("#aDiv").off('click');
```

可以看到，jQuery 新的事件 API 极大地简化了 jQuery 处理事件所需编写的代码。新的事件 API 同样具有强大的功能，而且使用起来更加便捷。它以一种更加直接、更加统一的方式实现了 jQuery 事件处理。

5.6 小结

经过本章的学习，你对 jQuery 脚本的掌握程度又提升到了一个新的高度。可以采用多种方式选择元素、遍历 DOM、操作任何元素的类和属性，还可以应用 jQuery 统一的事件处理机制，不必担心不同浏览器之间的差异。

本章还介绍了不同浏览器在事件处理问题上存在的历史差异，介绍了事件传播、Event 对象。jQuery 为开发人员剪除了这些苦恼，它提供了一种简单的解决方案，帮助开发人员轻松地管理事件。最后，本章还介绍了一个有用的示例，其中融合了本章介绍的事件的各种知识。

在第 6 章中，将使用事件的相关知识来处理数据和表单，并学习如何通过 Ajax 来提交表单。Ajax 是现代 Web 应用程序开发的基石。

5.7 参考

https://developer.mozilla.org/en/Gecko_DOM_Reference/
http://msdn.microsoft.com/en-us/ie/ff468705#_IntroEnhancedDOM
http://en.wikipedia.org/wiki/DOM_events#Event_handling_models
<http://www.quirksmode.org/js/dom0.html>
<http://www.w3.org/TR/DOM-Level-3-Core/>
<http://catcode.com/domcontent/events/capture.html>
http://www.quirksmode.org/js/events_order.html
<http://www.javascriptkit.com/jsref/event.shtml>
<http://stackoverflow.com/questions/4579117/jquery-live-vs-delegate>
<http://jquerybyexample.blogspot.com/2010/08/bind-vs-live-vs-delegate-function.html>
http://en.wikipedia.org/wiki/Internet_Explorer
<http://api.jquery.com/on/>



第 6 章

HTML 表单、数据和 Ajax

本章内容

- jQuery 数据应用程序
- 使用 jQuery 验证表单
- 使用 jQuery 的 Ajax 方法

本章介绍三个关系非常密切的主题。HTML 表单包含了数据，处理数据是 Web 开发的日常工作。通常情况下，可以对表单的数据进行处理，并以异步方式提交给一个服务器，而不再采用旧式的“提交——处理——重新加载页面” (post-process-reload) 技术。本章将介绍过去 10 年以来最具革命性的 Web 开发技术——Ajax。在 jQuery 使用 Ajax 技术非常简单，它极大地简化了客户端与服务器之间的异步通信。

6.1 jQuery 数据应用程序

使用诸如 alt 或 class 这样的属性将一些有用的数据附加到元素上，这是一种常见的错用标记元素功能的情形。例如：

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.6.1.js"></script>
    <script type="text/javascript">
      $(function(){
        $("#randomEl" ).attr( "alt" , "1999" );
      });
    </script>
```



```

</head>
<body>
  </div>
</body>
</html>

```

必须说明的是，我们知道可以在标记代码中直接把 `alt` 属性的值设置为 1999，但在这里我们假定需要以编程方式为 `` 标记代码添加 `alt` 属性和值。这种方式在语义上存在一定的问题，因为用于存储数据的属性与数据之间并没有语义上的联系。比如，1999 是一个年份、一个随机整数或表示其他什么含义？我们无从得知。更好的办法是使用 jQuery 的 `.data()` 方法，它可以在元素中存储任意数据，并为之定义一个有意义的键。`.data()` 方法的语法如下所示：

```
$.data( element, key, value );
```

可以采用下面的语法取回存储的数据：

```
$.data( element, key );
```

例如：

```

<!DOCTYPE html>
<html>
  <head>
    <script src=""></script>
    <script type="text/javascript">
      $(function(){
        $("#randomEl").data( "itemPrice", "1999" );
      });
    </script>
  </head>
  <body>
    </div>
  </body>
</html>

```

这样一来，显然 1999 不是一个年份，而是一个价格。HTML 5 引入了用户自定义数据属性，任何以 `data-` 为前缀的属性都可以在标准化的、以编程方式访问的脚本代码中使用。例如，下面的 `div` 元素包含了数据属性 `data-age`。这是一种将信息存储在客户端的替代方案。

```
<div id="person" data-age="31"></div>
```

Microsoft 为社区贡献了三个附加插件，并一度得到了 jQuery 项目的官方支持，这三个插件是：Data Link、Template 和 Globalization。本书第 11 章将介绍 Template 插件。下面先介绍一个 Data Link 插件，Data Link 可以在对象之间实现双向绑定(two-way binding)。也就是说，它可以将一个对象的字段连接到另外一个对象的字段，因此，改变第一个对象中某个字段的值，则第二个对象中相应字段的值也会随之改变。可以使用该技术来简化表单

与对象之间的通信，因为它消除了对象与表单进行数据交互的代码。比如下面的例子：



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.6.1.js"></script>
    <script src="jquery.datalink.js"></script>
    <script type="text/javascript">
      $(document).ready(function() {
        var registration = {};
        $( "form" ).link(registration);

        //设置一些默认值
        $(registration).setField( "name", "New User Registration");
        $(registration).setField( "email", "i_dont_exist@mail.com");

        $("form").change(function() {
          console.log( registration.name + " " + registration.email);
        });
      });
    </script>
  </head>
  <body>
    <form method="post">
      <label for="name">User Name </label>
      <input type="text" name="name" />

      <label for="email">email </label>
      <input type="email" name="email" />

      <input type="submit" value="send it" />
    </form>
  </body>
</html>
```

代码片段 datalink.txt

在上面的例子中，在修改了表单中的值之后，`console.log` 语句将输出 `registration` 对象所包含对象的值，它反映了表单中字段值的新变化。

可以看到，jQuery 提供了一些处理数据的快捷方式和方法，但是终端用户都是不可信的！下一小节将介绍数据验证的来龙去脉。

6.2 使用表单验证

出于很多原因，表单验证是非常重要的。它可以避免无意间弄乱数据库，保护系统免于恶意的注入攻击，它还有很多常见的作用，比如确保输入的数据是有效的。例如，在一

个姓名字段中，或许不应该允许输入数字或\$%&#@之类特殊字符。

传统的做法是使用 jQuery 或一些 JavaScript 表单，限制输入的数据格式，或者检查输入数据是否有效。最简单的例子就是检查一个要求必须输入数据的字段是否输入了数据，比如下面的代码：

```
if($("#myInput").val() !== ''){
    //处理代码
}
```

请注意，如果终端用户关闭了浏览器的 JavaScript 功能，那么精心编写的 JavaScript 数据验证代码就会失效。在保存数据之前，不但要在客户端验证数据，而且还应该在服务器端进行验证。

使用 Modernizr 库进行特性检测

在进一步介绍 HTML5 表单和 jQuery 之前，最好先学习一下 Modernizr 库。它可以极大地简化检测浏览器特性的任务，这些特性可能仍未被所有主流浏览器实现。Modernizr 库通过一个名为“特性检测”的过程来实现这种功能。Modernizr 库可以取代旧式的 User Agent (UA)探测方法，这种方法根据一个不可信的 navigator.userAgent 字符串来检测浏览器的功能。特性检测则是测试特定的浏览器功能是否可用。特性检测方法更能适应未来的变化、对用户也更加友好。采用特性检测方式，即使出现了一些新的浏览器、或者某些 Chrome 或 Firefox 的版本为了赢得 Web 的青睐而使用了一些异常的 navigator.userAgent 字符串，这无所谓。如果浏览器支持指定的特性，Modernizr 库就会报告该特性有效。

例如，可以使用 Modernizr 库来检测某个浏览器是否支持 canvas 元素。在下面的例子中，可以看到 Modernizr 库一个非常优秀的功能。对于每一个要检测的特性，它在 Modernizr 对象中保存一个相应的布尔类型值作为标记，以反映特性测试的结果。对于下面的例子，如果浏览器支持 canvas 元素，则 Modernizr.canvas 的值为 true。除了执行特性检测之外，Modernizr 库还会在 HTML 元素中添加一些 CSS 类，以指示特性的可用性。



```
<!DOCTYPE html>
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.6.1.js"></script>
  <!-- Link to modernizr, useful for feature detection -->
  <script
    src="http://ajax.cdnjs.com/ajax/libs/modernizr/1.7/modernizr-1.7.min.js">
  </script>

  <script type="text/javascript">
    if(Modernizr.canvas){
      //在这里编写依赖于 canvas 元素的代码
    } else {
      //相反地，如果 canvas 不存在，则在这里编写处理这种情况的代码
    }
  </script>
```



```

</script>
</head>
<body>
</body>
</html>

```

代码片段 modernizr.txt

在任何需要使用 Modernizr 库的地方，都可以使用公开的 JavaScript CDN：ajax.cdnjs.com。下面将演示一下 Modernizr 库的实际应用。HTML5 还提供了另外一个重要的特性，就是在字段中添加 `required` 标志，就可以将该字段定义为一个必填字段。比如下面的表单：

```

<!DOCTYPE html>
<html>
<body>
<form>
<input type="text" name="userName" required/>
<input type="password" name="password" required/>
<input type="submit" value="send it" />
</form>
</body>
</html>

```

在支持 `required` 标志的浏览器中，该表单将呈现为图 6-1 所示的页面。在本例中，使用的是 Chrome 浏览器，它显示了在试图提交一个空表单时产生的效果。

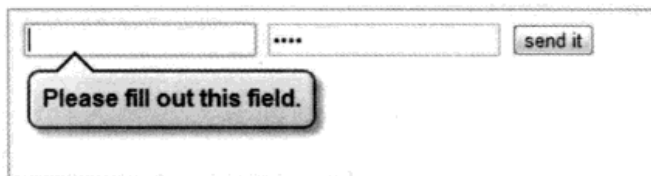


图 6-1

并非所有浏览器都支持 `required` 这一属性，或者支持 HTML5 的其他新特性。jQuery 再次发挥威力，简化了验证表单的工作。在下面的代码中，首先使用特性检测判断浏览器是否支持 `required` 字段。如果浏览器不支持 `required` 字段则使用 jQuery 来实现必填字段的功能。Modernizr 库将检测是否存在 `required` 字段特性，如果不存在，则将一个用于显示消息的 `div` 元素追加到文档中，并将一个简单的表单 `submit` 事件处理程序附加到表单上。在该事件处理程序中，将使用 jQuery 的表单工具函数 `val()` 检查每一个必填字段是否为空。如果有任何一个必填字段为空，则将消息 `div` 显示出来，为空字段添加一个红色的实线边框，并阻止表单提交。这是一个非常简单的表单验证的例子，但是它说明了使用 jQuery 执行表单验证的基本概念，以及使用特性检测来利用现代浏览器功能的基本概念。



```

<!DOCTYPE html>
<html>

```

可从
Wrox.com
下载源代码

```

<head>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script src="http://ajax.cdnjs.com/ajax/libs/modernizr/1.7/modernizr-1.7.min.js"> </script>

  <script type="text/javascript">
    $(function() {
      if( !Modernizr.input.required ){
        var $msg = $("<div id='reqMessage'>Required Fields Missing</div>");
        $msg.css("background-color", "yellow")
          .hide();

        $("body" ).append($msg);

        $("#fDet" ).on("submit", function(e) {
          $("input[required]").each(function() {
            if ( $(this).val() === "" ) {
              $("#reqMessage").show();
              $(this).css("border" , "1px solid red");
              e.preventDefault();
            }
          });
        });
      }
    });
  </script>
</head>
<body>
<form id="fDet" action="#">
  <input type="text" name="userName" required/>
  <input type="password" name="password" required/>
  <input type="submit" value="send it" />
</form>
</body>
<html>

```

代码片段 featureDetection-form.txt

6.3 使用 HTML 表单元素

从上面的例子中可以看到，在不支持 HTML5 的浏览器之中，可以使用 jQuery 技术来重新构造相应的 HTML5 功能。这一小节将介绍 HTML5 中新引入的一些表单元素。

什么时候不必使用 jQuery，什么时候需要使用 jQuery，搞清楚这一问题非常重要。HTML5 引入了一些新的表单元素和一些新功能。表 6-1 列出了一些精选的新 input 类型和控件。

表 6-1 HTML5 新的表单控件

控 件 类 型	INPUT 类型	描 述
URL	url	一个用于编辑 URL 的控件
E-mail	e-mail	一个用于输入或编辑电子邮件地址的控件
Telephone	tel	一个单行的纯文本编辑控件, 用于输入一个电话号码
Search	search	一个单行的纯文本编辑控件, 用于输入一个或多个搜索项
Slider	range	一个滑块控件, 用于将元素的值设置为一个表示数值的字符串
Number	number	一个可精确设置数值的控件, 用于将元素的值设置为一个表示数值的字符串
Date	date	日期控件, 用于将元素的值设置为一个表示日期的字符串
Date and Time	datetime	日期时间控件, 用于将元素的值设置为一个表示全球日期和时间的字符串(带有时区信息)
Color	color	一个颜色拾取控件, 用于将元素的值设置为一个表示简单颜色的字符串

请参考 <http://www.w3.org/TR/html-markup>

除了新的 input 控件之外, HTML5 还引入了一个用于添加占位文本的快捷属性。占位文本是在输入字段中显示的一个临时的默认文本, 当用户将输入焦点移到该字段上时, 占位文本就会消失。在 HTML5 之前, 需要使用 JavaScript 来实现这种效果, 但较新的 HTML5 浏览器默认使用一个名为 placeholder 的属性来支持该功能。例如, 下面的代码示例显示了两个输入框, 并分别为每一个输入框设置了 placeholder 属性。图 6-2 显示了这两个输入框在页面上呈现的效果。

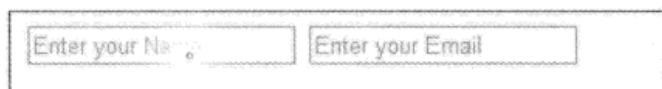


图 6-2

```
<!DOCTYPE html>
<html>
  <body>
    <form>
      <input type="text" placeholder="Enter your name" >
      <input type="text" placeholder="Enter your email" >
    </form>
  </body>
</html>
```

在浏览器的层面上, placeholder 属性实现了通常需要硬编码才能实现的功能增强。placeholder 属性很好用, 不必为它编写任何一行代码。但遗憾的是, 并非每一种浏览器都天然地支持 placeholder 属性, 因此在实际的应用中还不能完全放弃 JavaScript 代码的解决

方案。下面的代码示例说明了如何使用 jQuery 来实现 placeholder 属性的功能。它首先使用 Modernizr 库来检测浏览器是否支持 placeholder 属性，然后编写了一些 jQuery 代码来弥补其他浏览器对 placeholder 属性支持的不足。

在产品化的环境中，开发人员也许会喜欢使用某种优秀的 placeholder 插件，比如 Mathias Bynens 编写的插件之一：<https://github.com/mathiasbynens/jquery-placeholder>。但是学习在不使用这些插件的情况下如何解决问题，仍然是非常有意义的。特性检测和 polyfill 是现代 Web 开发的坚强后盾，因此研究一下 polyfill 底层的工作原理，要比简单使用插件更有启发性。



```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
    <script src="http://ajax.cdnjs.com/ajax/libs/modernizr/1.7/modernizr-1.7.min.js">
    </script>
    <script type="text/javascript">
      $(function(){
        if( !Modernizr.input.placeholder ){
          $('input[type=text]') .focus(function(){
            if( $(this).val() === $( this ).attr( 'placeholder' ) ){
              $(this).val('');
            }
          });
          $('input[type=text]') .blur( function(){
            if( $( this ).val() === '' ){
              $( this ).val( $( this ).attr( 'placeholder' ) );
            }
          });
        }
      });
    </script>
  </head>
  <body>
    <form>
      <input type="text" name="userName" placeholder="Enter your Name" />
      <input type="text" name="address" placeholder="Enter your Address" />
    </form>
  </body>
</html>
```

代码片段 placeholder.txt

下面的例子实现了一个电子邮件输入的小组件。类似地，可以编写一个 HTML5 的原生版本，但无法获得所有浏览器的支持。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
    <script
      src="http://ajax.cdnjs.com/ajax/libs/modernizr/1.7/modernizr-1.7.mi
      n.js">
    </script>

    <script type="text/javascript">
      $(function(){
        var emailRegEx = /^([\w-\.]@([\w-]+\.)+[\w-]{2,4})?$/;
        $( "#errorDiv" ).hide();
        If ( !Modernizr.inputtypes.email ){
          $( "input[type=email]" ).blur( function(){
            var emailValue = $( this ).val();
            if( emailValue !== '' ){
              var passes = emailRegEx.test( emailValue );
              if( !passes ){
                //显示验证错误消息
                $( "#errorDiv" ).show();

                //禁用 submit 按钮
                $( "input[type='submit']" ).attr( "disabled" );
              } else {
                $( "#errorDiv" ).hide();
                $( "input[type='submit']" ).attr( "disabled","" );
              }
            }
          });
        }
      });
    </script>
  </head>
  <body>
    <form>
      <input type="email" name="companyEmail" />
      <input type="submit" />
      <div id="errorDiv">Invalid email format.</div>
    </form>
  </body>
</html>
```

代码片段 email-input.txt

HTML5 控件可以优雅地回退。如果浏览器不支持特定的 HTML5 控件——比如在本书写作时，只有 Opera 浏览器支持 color 控件——则它将显示为一个普通的文本输入框。

在某些情况下，我们可能想实现这样的验证：将输入的值与数据库中的值进行比较，使用 Ajax 就可以实现这样的效果。在下面的小节中，先简要地介绍一下 Ajax 技术，然后我们再回到表单验证的问题。

6.4 Ajax 基础

Ajax 这一术语，最初的定义是 Asynchronous JavaScript and XML 这几个单词的首字母缩写。它是一种在服务器之间异步地发送和接收数据(使用 XML)，而不必完全重新加载整个 Web 文档的方法。Ajax 技术不断演化。从前面的章节中可以看到，现在 Ajax 指的是在客户端与服务器之间采用异步通信方式、使用 JavaScript 创建交互式 Web 和动态应用程序的一种开发方法。近年以来，人们不再总是使用 XML 作为服务器与客户端之间传递数据的格式，而是更多地采用其他格式，比如 JSON。

Ajax 技术的核心是 XMLHttpRequest 对象，它可以追溯到 2000 年，当时 Microsoft 为了实现服务器与客户端的异步通信而引入了 ActiveX 对象。在 2005 年，Jesse James Garrett 创造了 Ajax 这一术语，用来描述这种异步通信技术。

不同浏览器在管理事件的方式上存在差异。与之类似，不同浏览器在使用 Ajax 技术时也有着差异。但是这种差异并不复杂。例如，要在 Internet Explorer 浏览器中创建一个新的请求对象，可以使用下面的代码：

```
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
```

在较低版本的 IE 中则使用下面的代码：

```
var xhr = new ActiveXObject("MSXML2.XMLHTTP");
```

在 IE7 或更高版本中，或者任何支持 XMLHttpRequest 的其他浏览器中，则可以直接使用原生的 JavaScript 对象 XMLHttpRequest：

```
var xhr = new XMLHttpRequest();
```

幸运的是，无论是使用标准的 XMLHttpRequest 对象，还是使用 Microsoft 的 ActiveX 对象，二者的代码非常类似，因此这大大减少了编码的麻烦。典型情况下，如果你想自己处理这种差异的细节问题，则可以使用下面的代码来实例化一个请求对象：

```
/*
支持较低版本 Internet Explorer 中的早期的 ActiveX 对象，可适用于直到 IE5 这样低的版本
*/
if ( typeof XMLHttpRequest == "undefined" ) {
    XMLHttpRequest = function () {
        try {
            return new ActiveXObject( "Msxml2.XMLHTTP.6.0" );
        }
        catch (e) {}
    }
}
```



```

try {
    return new ActiveXObject( "Msxml2.XMLHTTP.3.0" );
}
catch (e) {}

try {
    return new ActiveXObject( "Msxml2.XMLHTTP" );
}
catch (e) {}
    throw new Error( "No XMLHttpRequest." );
};
}
var xhr = new XMLHttpRequest();

```

在处理好创建请求对象的细节之后，还必须使用 `xhr.open()` 方法与服务器建立一个连接，该方法的语法如下所示：

```
xhr.open(method, url, isAsynchronous);
```

在 `xhr.open()` 方法中，第一个参数 `method` 指的是 HTTP 的方法，即在指定资源上执行的行为。虽然 HTTP 方法(或称为“动词”)的完整列表非常长，但典型情况下 Ajax 请求只使用两种 HTTP 方法：POST 或 GET。第二个参数 `url` 是用于交换数据的服务器 URL 地址。第三个参数 `isAsynchronous` 是一个布尔标志，用于指示该调用是否采用异步方式。该标志指出该请求是否应该阻塞其余脚本的执行。Ajax 技术这一名称暗示了采用异步方式，因此通常将 `isAsynchronous` 参数设置为 `true`。也可以设置请求对象采用同步方式与服务器进行通信，但这违背了 Ajax 技术的初衷。在打开连接之后，使用 `xhr.send()` 方法初始化到服务器的请求。该方法接收一个 `data` 参数：

```

xhr.open("POST", "/request/url/", true);
xhr.send(data);

```

由于 Ajax 的通信是异步方式，因此无法按照顺序方式来编写要执行的方法。例如：

```

xhr.send(data);
doSomething();

```

由于 `send()` 方法执行后将立即返回，因此请求和响应这两个操作分别属于两个独立的线程。在上面的代码中，当执行 `doSomething()` 时，还没有获得来自服务器的响应。解决这一问题的办法是使用 `onreadystatechange` 事件和回调函数，当一个请求的状态发生改变时将调用该回调函数。因此，可以将之前的代码块改写为：

```

xhr.onreadystatechange = function(){
    if( this.readyState === 4 ){
        if( this.status >= 200 && this.status < 300 ){
            doSomething();
        } else {

```

```

        //通信存在问题
    }
}
xhr.send( data );

```

这样一来,当请求完成并成功时,将执行 doSomething()方法。表 6-2 列出了不同的 readyState 状态值。

表 6-2 ReadyState 属性

状 态	描 述
0: 未初始化	XMLHttpRequest 对象已经构造完成
1: 正在载入	已经成功调用了 Open()方法。在该状态期间,可以使用 setRequestHeader()方法设置请求的头部,可以使用 send()方法发送请求
2: 载入完成	所有重定向(如果有的话)都已经完成,并且已经接收到了最终响应的全部 HTTP 头。XMLHttpRequest 对象的一些响应成员已经可用
3: 交互	正在接收整个响应体
4: 完成	数据传输已经完成,或者在传输期间发生了错误(比如无限重定向)

请参考 <http://www.w3.org/TR/XMLHttpRequest/#states>

在请求完成并成功后,可以使用 xhr 对象的 responseText 属性获取服务器响应的数据。responseText 属性返回什么样的数据,取决于服务器端的代码。可以是文本或任何格式,而不仅仅是 XML。就目前而言,从服务器可以返回多种数据格式——HTML、XML、JSON、纯文本等等,对于解析返回的数据而言,每一种格式都有自己的利弊。

近年以来,主流的趋势是使用 JSON 作为 Ajax 数据交换的格式。由于它是 JavaScript 原生对象,因此它更容易以编程方式进行处理。使用 JSON 格式的数据时,不必采用 DOM 遍历方法来搜索服务器返回的 XML DOM,只需要简单地使用点号表示法就可以访问 JSON 响应中的属性。另外,JSON 不像 XML 那么冗长,因此对于每一个请求,它所占用的网络带宽更小。

当然,你还会看到 XML、使用纯文本甚至是返回整块 HTML 代码的情况,但近年以来使用 JSON 格式已经越来越普遍。

毋庸置疑,jQuery 以一种更加简单的方式,提供了处理 Ajax 请求和响应的新方法。

6.5 在 jQuery 中使用 Ajax

对于事件处理,jQuery 抽象了事件的细节。对于 Ajax 也是如此,它抽象了 Ajax 通信的大量细节问题。例如,在 jQuery 中开发人员不必担心 xhr 对象是如何实例化的。在 jQuery 中用于发送 Ajax 调用的通用方法是 \$.ajax()方法,它的语法为:

```
$.ajax(url, [settings]);
```

或者只发送一个 settings 对象：

```
$.ajax([settings]);
```

在下面的一些例子中,要求具备一个服务器以响应 Ajax 请求。可以采用 Python 或 PHP 作为后台服务器,以支持前端的 Ajax 请求。遗憾的是,Python 或 PHP 超出了本书的内容范围。

表 6-3 列出了一些\$.ajax()方法中最重要的设置。完整的设置列表请参考在线文档 <http://api.jquery.com/jquery.ajax/#jQuery-ajax-settings>。

表 6-3 jQuery 中\$.ajax()的设置

属 性	描 述
url	所请求资源的 URL
type	Ajax 请求的 HTTP 方法,通常是一个 GET 方法或 POST 方法。根据浏览器的支持,其他较少使用的 HTTP 方法,比如 PUT 或 DELETE 方法也可能用到
data	要发送给服务器的数据
dataType	希望从服务器返回的数据类型。有效的数据类型包括: "XML"、"HTML"、"script"、"JSON"、"JSONP"和"text"
success(data, textStatus, jqXHR)	请求成功时调用的函数
error(jqXHR, textStatus, errorThrown)	请求失败时调用的函数
complete(jqXHR, textStatus)	当 Ajax 请求完成时调用的函数
timeout	为请求设置一个超时时间(以毫秒为单位)

下面的例子将向 URL 为/post/2/here.html 的页面发起一个请求,它接收一些返回的纯文本,如果请求成功则输出"made it"字符串,请求发生错误则输出"Something didn't work"字符串:

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script type="text/javascript">
    $.ajax({
      url : "/post/2/here.html",
      dataType: "html",
      success: function(r){
        console.log( "made it" );
      },
      error : function(r){
```



```

        console.log( "Something didn't work" );
    }
})
</script>
</head>
<body>
</body>
</html>

```

返回的有效数据类型包括：XML、HTML、JSON、JSONP、text 和 script。

在前面的章节中可以看到，jQuery 非常善于遍历 DOM 树。在服务器的响应是 XML 的情况下，jQuery 提供了 \$.find() 方法用于从 XML 文档中获取数据。假定在一个 Ajax 调用中，返回了一个描述一本图书的 XML 文档，我们想从中获取章节的标题。下面的代码说明了如何使用 \$.find() 方法来解析某些 XML 数据，并将其追加到当前文档的 DOM 中。



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
<head>
  <scriptsrc="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script type="text/javascript">
    $.ajax({
      url : "/book.html",
      dataType: "xml",
      success: function( data ){
        $( data ).find( "chapter" ).each(function() {
          $( "document" ).append($( this ).find( "title" ).text());
        });
      },
      error : function(data){
        console.log( "unable to process request" );
      }
    });
  </script>
</head>
  <body>
</body>
</html>

```

代码片段 ajax-find.txt

上面的例子将把从返回的 XML 数据中查找到的章节标题追加到文档的 <body> 元素之中。\$.find() 方法的功能是在一个 XML 文档中搜索具有指定元素名的元素。与 HTML 文档类似，如果想访问元素的属性，请使用 .attr() 方法。

近年以来，请求 JSON 格式的数据已经非常普遍，从逻辑上完全可以将 JSON 对象视为 JavaScript 对象。比起 XML 数据，JSON 对象的另外一个优势是它更轻量级。可以按照下面的方式，采用 \$.ajax() 方法从服务器获取 JSON 对象：

```
$.ajax({
  url: "/jsonBook.html",
  dataType: 'json',
  data: data,
  success: function( data ){
    //处理 json 数据
  }
});
```

jQuery 提供了一个快捷方法\$.getJSON(), 它以异步方式从服务器获取 JSON 格式的数据:

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
    <script type="text/javascript">
      $.getJSON("/url/2/get/json.php", function(data, text, jqXHR){
        //在这里处理回调函数中的任务
      });
    </script>
  </head>
  <body>
  </body>
</html>
```

遗憾的是, jQuery 没有提供对应的\$.postJSON()方法。

你可能已经注意到, 很多应用程序中的\$.ajax()方法都使用了一组常规的设置选项。jQuery 提供了一个名为\$.ajaxSetup()的方法, 可以以全局方式设置 Ajax 的选项。除非重新设置这些选项, 否则这些选项值将应用于任何一个\$.ajax()调用。\$.ajaxSetup()方法的语法为:

```
$.ajaxSetup(options);
```

其中, options 是一个对象实例, 该对象实例的各个属性用于设置全局 Ajax 属性。例如, 下面的代码设置了默认的 type、dataType 和 error 回调函数:

```
$.ajaxSetup({
  type : "GET",
  dataType : "json",
  error : universalGoofupFunction
});
```

请注意, \$.ajaxSetup()方法的全局设置将覆盖当前页面上所有 Ajax 请求的设置。因此, 除非你完全控制了页面中运行的所有 JavaScript, 否则在使用\$.ajaxSetup()时应该谨慎小心。

在继续介绍 Ajax 技术之前, 先来看一下 jQuery1.5 中引入的一个新特性, 即延迟对象(deferred object)。延迟对象是 promise 模式的一个例子, 一个 promise 就是一个“承诺在将

来某个方便的时候提供所需值”的对象。对于异步方式来说，这非常有用。jQuery 的延迟对象也是基于 `promise` 模式的，它允许链式调用回调函数。

延迟对象开始的状态是 `unresolved`，它的状态可以从 `unresolved` 转变为 `resolved` 或者 `rejected`。延迟对象将在第 13 章深入介绍。

`$.ajax()` 方法，实际上 jQuery 的所有 Ajax 方法，都会返回一个 `jqXHR` 对象，它是 `XmlHttpRequest` 对象的一个超集。它允许将 Ajax 调用重写为链式调用方式：

```
$.ajax({url : "/go/here"})
    .success(function(){ /* handle success */ })
    .complete(function(){ /* handle complete method */ });
```

此外 jQuery 还提供了一个快捷的工具方法，比如 `$.get()` 方法和 `$.post()` 方法，分别用于发送 `get` 请求和 `post` 请求。对于下面的代码：

```
$.ajax({
    type: "get",
    url: "/go/here.php",
    data: data,
    success: callBackFunction,
    dataType: "json"
});
```

可以使用 `$.get()` 方法改写为：

```
var jqxhr = $.get( "/go/here.php", function(data){} );
```

`$.get()` 方法的通用语法为：

```
$.get( url, parameters, callback, type );
```

与 `$.ajax()` 方法类似，`$.get()` 方法也返回一个 `jqXHR` 对象。对应 `post` 请求的 `$.post()` 方法，除了请求类型不一样之外，其他参数与 `$.get()` 方法完全相同。`$.post()` 方法的语法如下所示：

```
$.post( url, parameters, callback, type );
```

jQuery 还具有一个极为有用的方法：`$.load()`。它的功能是发起 Ajax 请求，并将返回的 HTML 片段加载到匹配的元素中。所有这些功能都在一次方法调用中实现。该方法的语法如下所示：

```
$( selector ).load( url, [data], [success(response, status, XHR)] )
```

`$.load()` 方法接收 3 个参数：第一个参数是一个目标 `url`，第二个参数是一个可选的数据对象，第三个参数是一个可选的 `success` 函数，该函数本身又接收一个响应字符串、一个状态字符串和一个 `XmlHttpRequest` 对象(`XHR`)作为参数。例如：

```
$("#latestNews").load("/getLatest.php",dateObj, function(resp, status,
XHR){
    if(status === "success"){
```



```
//执行处理
}
});
```

如果服务器上所请求的页面工作正常,那么 latestNews 元素将加载从服务器返回的文本。

本章的最后一个例子是一个订购披萨的表单,其中使用了 HTML 5 的新控件和 jQuery 的 \$.load() 方法。为了兼容使用较低版本浏览器的用户,将使用 jQuery 来实现 HTML 5 新控件的功能,另外还使用 jQuery 来验证输入和提交表单。

请注意,在这个例子中并未处理用户禁用 JavaScript 的情况。在设计 Web 表单时,这是一个应该考虑的重要因素,妥善解决这一问题的很多工作(即在服务器端建立一个回调验证服务)超出了本书的范围。

用户不必先在披萨店注册就可以订购披萨,但在订购时必须填入一些必要的信息。订购披萨的表单具有下列字段:

- first name
- last name
- address
- state
- zip
- phone
- e-mail
- time of delivery
- cc type
- cc number
- pizza size
- toppings

首先是设计订购表单,在表单中使用 placeholder 文本来取代<label>作为每一个字段的标题。除了 toppings 字段之外,其他所有字段都是必填字段。state 字段将使用 Ajax 的 autocomplete 特性。为了嗅探浏览器的特性,再次使用了 Modernizr 库。为了让例子更加完美,添加了 Data Link 插件以简化从表单中获取各个字段数据并发送给服务器的处理过程。在将表单提交给服务器之后,如果没有抛出异常,将向用户显示一个订单的发票。

下面就是该页面的标记代码。可以从本书网站下载到完整的源代码,不必手工输入!

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
    <script
      src="http://ajax.cdnjs.com/ajax/libs/modernizr/1.7/modernizr-1.7.min.js">
    </script>
```

```

<script src="jquery.datalink.js"></script>
<script type="text/javascript">
    //应用程序代码放在这里
</script>
</head>
<body>
    <h1>Enter Order Information</h1>
    <form id="pizzaOrderForm">
        <input type="text" name="firstName" placeholder="First Name" required
    >
        <input type="text" name="lastName" placeholder="Last Name" required >
        <input type="text" name="address" placeholder="Address" required >
        <input type="text" name="state" placeholder="State" required >
        <input type="text" name="zip" placeholder="Zip Code" required >
        <input type="tel" name="phone"
            pattern="999-999-9999" placeholder="Phone" required >
        <input type="email" name="email" placeholder="Email" required >
        <input type="text" name="timeOfDeliver" placeholder="Time to Deliver"
            required >
        <select name="ccType">
            <option value="visa">Visa
            <option value="amex">AmEx
            <option value="discover">Discover
        </select>
        <input type="text" name="ccNumber" placeholder="CC Number" required >
        <input type="text" name="pizzaSize" placeholder="" required >
        <select name="pizzaSize">
            <option value="0">Pick a size
            <option value="1">personal
            <option value="2">small
            <option value="3">medium
            <option value="4">large
            <option value="5">sportsman
        </select>
        <label> Number of Pizzas
            <input type="number" name="numOfPizzas" min="0" max="99"
                value="0" placeholder="Zip Code" required ></label>
        <label>Pepperoni<input type="checkbox" name="toppings"
            value="pepperoni"></label>
        <label>Garlic<input type="checkbox" name="toppings"
            value="garlic"></label>
        <label>Mushroom<input type="checkbox" name="toppings"
            value="mushrooms"></label>
        <label>Sausage<input type="checkbox" name="toppings"
            value="sausage"></label>
        <input type="button" value="Order!" >
    </form>
    <div id="price"></div>
</body>
</html>

```

与本章前面介绍的例子一样，该示例使用 Modernizr 库来检查浏览器是否支持 type 为 email 的<input>元素、required 属性和 placeholder 特性。在下面的 JavaScript 代码中可以看到，当其中任何一项特性缺失时，将使用 jQuery 来弥补它的功能。

为了自动填充 order 对象，使用了 Data Link 插件，从而简化了一些中间处理步骤。使用 Data Link 插件很简单，只需要创建一个空对象作为表单字段数据的容器即可。然后选中表单、并在包装对象上调用 link()方法即可。

```
var order = {};
$("#pizzaOrderForm").link(order);
```

在该例中，使用了传统的表单提交方法，并添加了一个简单的测试，以检查在不支持 required 属性的浏览器中，必填字段中是否已经输入了数据。\$.load()方法用于将订单提交给服务器。从服务器返回的响应文本是一个 HTML 字符串，\$.load()方法将把该 HTML 字符串插入到 id 为#price 的 div 元素之中，这样就不必编写将 HTML 文本追加到文档中的代码。下面的代码片段就是完成后的 JavaScript 示例。

```
$("#form").bind( "submit",function(e) {
    var valid = true;
    if(!Modernizr.input.required){
        $("#input[required]").each(function() {
            if ($(this).val() === "" ) {
                $("#reqMessage").show();
                $(this).css("border" ,"1px solid red");
                valid = false;
            }
        });
    }
    e.preventDefault();
    if (valid) {
        $("#price").load( "/process", data, function(responseTxt, status,
        XHR ){
            if(status === "success"){
                $("[type=button]").attr("disabled");
            } else if(status === "error"){
                $("#price").append("unable to process request, game over man");
            }
        });
    }
});
```



```
...
<script type="text/javascript">
    $(function(){
        //对于还不支持 HTML 5 的浏览器，使用 jQuery 来实现 placeholder 属性的功能
        if(!Modernizr.input.placeholder){
            $('input[type=text]').each(function() {
```



```

        $( this ).val( $( this ).attr('placeholder') );
    }); $('input[type=text]').focus(function(){
        if( $( this ).val() === $( this ).attr('placeholder') ){
            $( this ).val('');
        }
    });
    $('input[type=text]').blur(function(){
        if( $( this ).val() === '' ){
            $( this ).val( $( this ).attr('placeholder') );
        }
    });
}

//支持 required 属性的浏览器实现
if(!Modernizr.input.required){
    var $msg = $("<div id='reqMessage'>Required Fields Missing</div>");
    $msg.css("background-color","yellow")
        .hide();
    $("body").append( $msg );
}

//email 输入框的实现
var emailRegex = /^[^(\w-\.\.]+\@([\w-]+\.\.)+[\w-]{2,4})?$/;
if( !Modernizr.inputtypes.email ){
    $('input[type=url]').blur( function(){
        var emailValue = $( this ).val();
        if( emailValue !== '' ){
            var passes = emailRegex.test(emailValue);
            if( !passes ){
                //display validation error message
                $( "#errorDiv" ).show();
                //disable submit button
                $( "input[type='submit']" ).attr( "disabled" );
            } else {
                $( "#errorDiv" ).hide();
                $( "input[type='submit']" ).attr( "disabled","" );
            }
        }
    });
}

//订单处理
var order = {};
$( "#pizzaOrderForm" ).link( order );
$( "form" ).bind( "submit", function( e ){
    var valid = true;
    if(!Modernizr.input.required){
        $( "input[required]" ).each(function(){
            if ( $( this ).val() === "" ) {
                $( "#reqMessage" ).show();
            }
        });
    }
});

```

```

        $( this ).css( "border" , "1px solid red" );
        valid = false;
    }
});
}
e.preventDefault();
if (valid) {
    $("#price").load("/process",data,function(responseTxt, status,XHR){
    if(status === "success"){
        $( "[type=button]" ).attr( "disabled" );
    } else if(status === "error"){
        $( "#price" ).append( "unable to process request, game over man" );
    }
    });
}
});
});
</script>
</head>
...

```

代码片段 pizza-form.txt

图 6-3 并排显示了该订单页面在两种不同浏览器之中的对比，左侧是支持 HTML5 的 Opera 浏览器，右侧是不支持 HTML5 的 Firefox 2 浏览器。

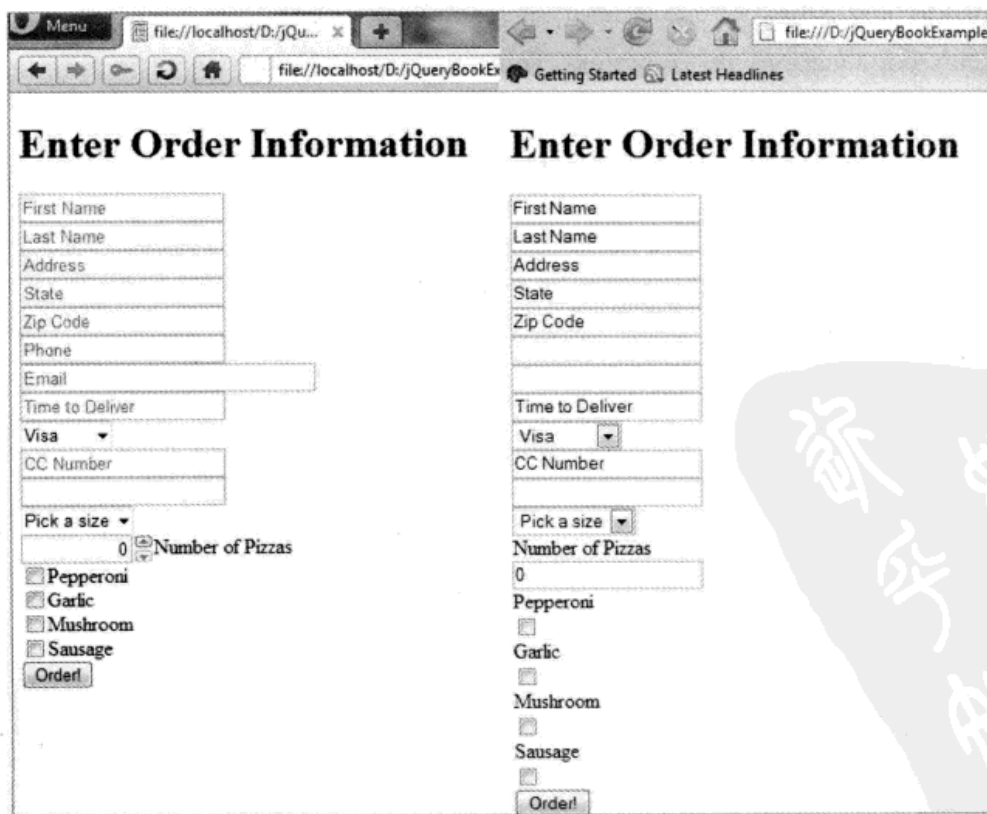


图 6-3

6.6 小结

本章介绍了开发人员需要掌握的大部分 Ajax 知识。可以看到, jQuery 有助于以一种语义化的方式将数据添加到元素中, 它还有助于执行表单验证、在目前仍不支持 HTML5 的表单中实现 HTML5 的功能。本章还介绍了 Ajax 的基础知识, 介绍了 jQuery 如何简化异步调用, 使 Ajax 的使用更加简单。jQuery 还提供了很多工具方法, 进一步简化了常用的 Ajax 操作, 比如\$.get()方法和\$.post()方法。

6.7 参考

<http://diveintohtml5.info/forms.html>
<http://www.bennadel.com/blog/1404-jQuery-Data-Method-Associates-Data-With-DOM-Elements-SWEET-ASS-SWEET-.htm>
<http://marcgrabanski.com/article/5-tips-for-better-jquery-code>
<http://www.matiasmancini.com.ar/jquery-plugin-ajax-form-validation-html5.html>
[http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))
<http://www.jibbering.com/2002/4/httprequest.html>
<http://www.w3.org/TR/html-markup/input.email.html>
[http://msdn.microsoft.com/en-us/library/ms536648\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms536648(v=vs.85).aspx)
http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
<http://ejohn.org/blog/html-5-data-attributes/>
<http://www.modernizr.com/docs/#features-html5>
<http://wiki.commonjs.org/wiki/Promises/A>
<http://www.erichynds.com/jquery/using-deferreds-in-jquery/>



第 7 章

动画和特效

本章内容

- 为元素和 CSS 属性创建动画
- 改变元素尺寸
- 创建用户自定义动画
- 使用 HTML5 的 canvas 元素创建动画

在 Web 上创建动画是一个很有趣的功能。这里所说的动画，不仅仅指的是在页面上移动元素，还包括切换元素的可视状态、在指定时间内连续地改变元素的属性，或者在对某个事件做出响应时修改元素的属性，以及改变元素的大小。大部分 Web 动画都是用 Flash 创建的，JavaScript 仅用于一些简单的情形，比如翻转图片。但是风水轮流转，在 jQuery 的帮助下，通过 JavaScript 创建 Web 动画越来越有优势。

本章将介绍 jQuery 为在 Web 应用程序中创建动画功能提供的一些快捷方法，比如移动、淡化元素、切换元素状态和改变元素大小等。本章还将介绍如何使用 `$.animate()` 方法，以并行方式联合多个不同的动画效果，最后还介绍了如何将这些技术应用于新的 HTML5 标记：canvas。

7.1 为元素创建动画效果

元素(比如 div)具有两个位置属性，用于确定它在 Web 页面上的位置：top 属性和 left 属性。元素的位置可以用百分比来表示，也可以用一个长度单位比如像素或厘米来表示。

下面的代码示例说明了动画最基本的底层概念，它以编程方式调整元素在页面上的位置。只要用户单击了文档，就会将绿色背景的 div 元素向右移动 10 个像素。

```

<!DOCTYPE html>
<html>
<head>
  <style type="text/css">
    div {
      position: absolute;
      left:10px;
      top:10px;
      width:20px;
      height:20px;
      background-color:green;
    }
  </style>
  <script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
  <script type="text/javascript">
    $(function(){
      $(document).click(function(){
        var pos = $("div").css("left");
        pos = parseInt(pos);
        $("div").css("left", pos+10);
      });
    });
  </script>
</head>
<body>
  <div>
</div>
</body>
</html>

```

可以使用\$.attr()方法来获取元素的 top 属性和 left 属性,并将其转换为数值类型。更好的办法是使用\$.offset()方法,它返回一个包含数值类型的 left 值和 top 值的对象,指出元素相对于文档的位置。调用\$.offset()方法的格式为:

```
$(selector).offset();
```

在下面的代码中,获取了 id 为 box 的一个 div 元素的偏移值,并将其 left 属性和 top 属性显示在一个警告信息框中:



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
<head>
  <style type="text/css">
    div {
      position: absolute;
      left:10px;
      top:50px;
      width:20px;
      height:20px;

```

```

        background-color:green;
    }
</style>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script type="text/javascript">
    $(function(){
        $("div#box").click(function(){
            var pos = $(this).offset();
            $("#watcher").text(pos.top+" "+pos.left);
        });
    });
</script>
</head>
<body>
    <p>Where is the box? <span id="watcher"></span></p><div id="box"></div>
</body>
</html>

```

代码片段 offset.txt

在上面这个例子中，当单击了 div 元素时，将显示该元素的坐标。另外，还可以使用 offset()方法来设置元素在页面上出现的位置：

```
$(selector).offset(coordinatesObject);
```

其中，coordinatesObject 是一个对象，它包含了一个 top 属性和一个 left 属性，并且这两个属性的值是整数。例如：

```

$("div#box").offset({
    top:100,
    left:100
});

```

在某些情况下，我们不想让元素相对于文档进行定位，而是想让它相对于它的父容器元素进行定位，此时可以使用 \$.position()方法。它设置和获取坐标的语法与 \$.offset()方法完全相同。

7.2 用 CSS 属性创建动画

很多简洁的特效都是在响应某个事件时，通过处理元素的 CSS 属性实现的，例如显示或隐藏一个元素。有两个不同的 CSS 属性可以控制一个元素在页面上是否显示，一是 display 属性；二是 visibility 属性。将 display 属性设置为 none，或者将 visibility 属性设置为 hidden，都可以隐藏一个元素。相反地，可以将 display 属性设置为 block，或者将 visibility 属性设置为 visible，都可以显示一个元素。

尽管这两个属性都可以“隐藏”一个元素，但应该指出的是，这两个方法对页面的流

具有完全不同的影响。对于将 `visibility` 属性设置为 `hidden` 的元素，将继续稳固地保留在文档的流中。文档中位于该隐藏元素之后的元素，将保持原来的位置偏移。而对于将 `display` 属性设置为 `none` 的元素，将完全从文档流中剥离出去。可以用下面的代码隐藏一个元素：

```
$("#div#message").click(function(){
    $(this).css("display", "none");
});
```

并用下面的代码显示该元素：

```
$("#div#message").ready(function(){
    $(this).css("display", "block");
});
```

更好的办法是使用 jQuery 的 `$.hide()` 方法和 `$.show()` 方法。在下面的例子中，使用 jQuery 在页面的顶部显示一个消息框。一旦用户单击了 `div` 元素，就隐藏该消息框，单击容器元素，该消息框就会再次显示出来。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
    #message {
        position: absolute;
        left:0;
        top:0;
        width:100%;
        height:20px;
        background-color:orange;
    }
    #container {
        position:absolute;
        left:0;
        top:0;
        width:100%;
        height:500px;
    }
</style>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script type="text/javascript">
    $(function(){
        $("#message").click(function(e){
            e.stopPropagation();
            $(this).hide();
        });
        $(document).click(function(){
            $("#message").show();
        });
    });
```

```

</script>
</head>
<body>
  <div id="message">
    Fill out our annoying survey! (click to dismiss)
  </div>
</body>

```

代码片段 showhide.txt

Stack Overflow 和 Twitter 二者都具有一个非常简洁的特性,即在顶部显示一个消息框,除了该消息框是以动画方式滑入这一重要的区别之外,该消息框与上面的例子的效果非常类似。为了创建滑入动画效果, jQuery 提供了 \$.slideDown() 和 \$.slideUp() 方法用于为 div 元素创建滑入、滑出效果。这两个方法的语法如下所示:

```

$(selector).slideUp([speed,] [easing,] [callback]);
$(selector).slideDown([speed,] [easing,] [callback]);

```

在 \$.slideDown() 和 \$.slideUp() 方法中,三个参数都是可选的。动画的速度、或者动画发生的持续时间,可以用一个表示毫秒值的数值来表示,也可以用一个 'fast' 或 'slow' 这样的字符串来表示。speed 参数为 'fast' 的动画,持续时间是 200 毫秒;而 speed 参数为 'slow' 的动画,持续时间是 600 毫秒。如果省略了 speed 参数(或称为 duration),则默认的 speed 值是 400 毫秒。\$.slideUp() 方法和 \$.slideDown() 方法都可以接收一个回调函数作为第三个参数,一旦 \$.slideUp() 或 \$.slideDown() 函数执行完毕,就会调用并执行该回调函数。

另外,在这两个方法中,还可以包含一个可选的 easing 参数(缓进函数),它指出在不同的时间点上动画运行的速度有多快。在本书写作时,该参数的默认值是 swing, 可选值为 linear。此外,还有一些简洁的 easing 插件,可以提供多种不同的 easing 选项。可以参考 <http://james.padolsey.com/demos/jquery/easing/> 网站。在 jQuery UI 中也包含了更多的 easing 函数。

在下面的例子中,使用 jQuery 的 \$.slideUp() 和 \$.slideDown() 方法,取代了之前的 \$.show() 和 \$.hide() 方法,改进了上面的例子:



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
<head>
  <style type="text/css">
    div {
      text-align: center;
      color: white;
      font-size: xx-large;
      position: absolute;
      left:0;
      top:0;
      width:100%;

```



```

        height:50px;
        background-color:orange;
    }
</style>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script type="text/javascript">
    $(function(){
        $("div#message").click(function(e){
            e.stopPropagation();
            $("div#message").slideUp('fast');
        });
        $(document).click(function(){
            $("div#message").slideDown('fast');
        });
    });
</script>
</head>
<body>
    <div id="message">
        Fill out our annoying survey! (click to dismiss)
    </div>
</body>
</html>

```

代码片段 *slideup-slidedown.txt*

jQuery 还提供了另外一个滑动特效：淡化。它包含了两个具体的函数：\$.fadeIn() 和 \$.fadeOut()。这两个函数的语法几乎与滑动函数的语法完全相同，它也具有 speed、easing 和一个可选回调函数等相同的概念。

```

$(selector).fadeIn([speed,] [easing,] [callback]);
$(selector).fadeOut([speed,] [easing,] [callback]);

```

7.3 改变元素的尺寸

你可能期望 jQuery 具有一个简单的工具方法，可以改变元素的尺寸，但实际上 jQuery 中没有这样的方法。\$.resize()方法是一个用于响应 onresize 事件的事件处理程序。除非使用\$.animate()方法，否则我们只能自己编写代码。

要改变一个元素的尺寸，只需要使用\$.css()方法改变它的 height 属性和 width 属性。例如，要缩小一个已有元素的宽度，可以使用下面的代码：

```

var w = $("#myElem").css("width");
w = parseInt(size.replace("px","")) - 10;
$("#myElem").css("width", w-10);

```

但是使用\$.css()方法在本质上并不理想，因为它的返回值是一个带有度量单位的字符

串, 比如像素。最好的办法是使用\$.height()和\$.width()方法, 这两个方法可以获取元素的高度和宽度, 这样一来改变元素尺寸的代码变得更加简单:

```
var w = $("#myElem").width() - 10;
$("#myElem").width(w);
```

7.4 设计用户自定义动画

前面介绍的动画方法都很优雅, 但对于底层的动画操作还有一个更加直接的方法: \$.animate()。它是 jQuery 提供的一个更通用的创建动画的方法, 它可以同时修改多个属性, 创建多个属性同时变化的动画效果。任何使用数值类型值的属性, 比如 top、left、height、width 和 opacity 属性, 都可以被\$.animate()方法操作。该方法可以接收 4 个参数: 第一个参数是一个对象, 它包含了要生成动画的属性; 第二个参数是动画的持续时间; 第三个参数是一个字符串, 它声明了动画要使用的缓进算法; 第四个参数是一个回调函数, 一旦动画执行完毕就会调用并执行该回调函数。

```
$(selector).animate( properties, [duration,] [easing,] [complete] )
```

在下面的例子中, 当单击文档时, 将为一个普通的蓝色 div 元素创建一个动画, 使它从左向右移动 100px。这个动画非常简单。但是请注意, 为了让动画能够正常工作, div 元素的定位方式必须是 relative、absolute 或者 fixed。



```
<!DOCTYPE html>
<html>
  <head>
    <style type="text/css">
      div {
        position: relative;
        left:0;
        top:0;
        width:10px;
        height:10px;
        background-color:blue;
      }
    </style>
    <script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
    <script type="text/javascript">
      $(function(){
        $(document).click(function(event){
          $("#div#box").animate({
            left: '+=100px'
          }, 200);
        });
      });
    </script>
  </head>
```

```

<body>
  <div id="box">
  </div>
</body>
</html>

```

代码片段 custom-animation.txt

可以在同一时刻修改多个属性的值，从而创建多个属性同时改变的动画。在下面例子创建的动画中，一个 UFO 将向右移动，同时具有淡出效果。元素淡出页面的效果，是通过最后将 opacity 属性设置为 0 来实现的。



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
<head>
  <style type="text/css">
    img#alien {
      position: relative;
      left: 0;
      background-color: blue;
    }
  </style>
  <script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
  <script type="text/javascript">
    $(function() {
      var winWidth = $(document).width();
      var duration = 1000;
      $(document).click(function(event) {
        $("#alien").animate({
          opacity: 0,
          width: 10,
          height: 10,
          left: '+=1000px'
        }, duration );
      });
    });
  </script>
</head>
<body>
  
</body>
</html>

```

代码片段 two-attributes.txt

在浏览器中加载该示例，可以看到 UFO 图片在移动的同时缩小并淡出页面的效果。你可以修改 left 和 duration 属性的值，试验不同的速度和效果。图 7-1 显示了该动画的变

化过程。

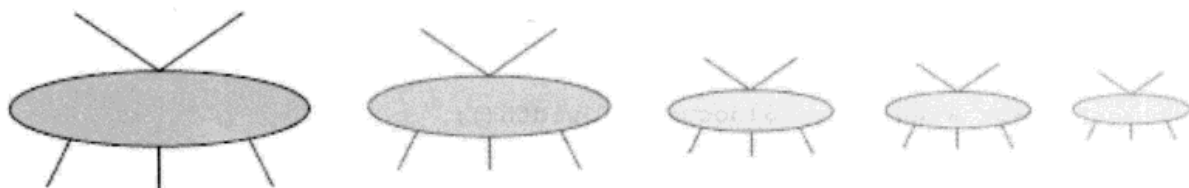


图 7-1

下面的例子比较长，但它说明了如何使用非常少的代码来实现大量的工作。该例子是一个类似于“太空入侵”风格的游戏的基础。在屏幕底部是一个外观非常原始的太空船，使用左箭头和右箭头键就可以控制太空船左右移动。按下箭头键将发射一道“激光”，它实际上是只是一个隐藏的 `div` 元素。在屏幕的顶部是一个外观凶恶的外星人，当激光击中它时它就从页面消失，即隐藏起来。

下面就是该示例的完整代码。其中包含了一些初始化的 CSS，以使例子能够正常工作。另外，还包含了少量必要的 JavaScript 代码，用于控制动画和交互性。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <style type="text/css">
      body {
        background-color:black;
      }
      h1 {
        color: yellow;
      }
      img#spaceShip {
        position: absolute;
        left:0;
        top:80%;
        background-color:blue;
      }
      img#invader {
        position: absolute;
        left:25%;
        top:10%;
      }
      div#laser {
        position: absolute;
        display: none;
        left:0;
        top:0;
        width:10px;
        height:70px;
        background-color: red;
      }
    </style>
  </head>
  <body>
```




```

    }
</style>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script type="text/javascript">
    $(function(){
        var winWidth = $(document).width();
        var duration = 1000;

        $(document).keydown(function(event){
            var keyCode = event.keyCode || event.which;
            var keyMap = { left: 37, up: 38, right: 39, down: 40 };

            switch (keyCode) {
                case keyMap.left:
                    $("#spaceShip").animate({
                        left: '-=50'
                    },200);
                    break;

                case keyMap.up:
                    var ufoLeft = $("#spaceShip").offset().left;
                    var ufoTop = $("#spaceShip").offset().top;
                    $("#laser").offset({left:ufoLeft+87, top:(ufoTop-30)});
                    $("#laser").css("display","block")
                        .animate({top:10}, 200, function(){
                            var invaderLeft = $("#invader").offset().left;
                            var laserLeft = $("#laser").offset().left;

                            if( laserLeft >= invaderLeft &&
                                laserLeft <= invaderLeft + 288){
                                $("#invader").hide();
                                $("body").html("<h1>Direct Hit!</h1>");
                            }

                            $("#laser").offset({left:0, top:0});
                            $("#laser").css("display","none");
                        });
                    break;
                case keyMap.right:
                    $("#spaceShip").animate({
                        left: '+=50'
                    },200);
                    break;
            }
        });
    });
</script>
</head>
<body>
    

```



```


<div id="laser"></div>
</body>
</html>

```

代码片段 *spaceInvader.txt*

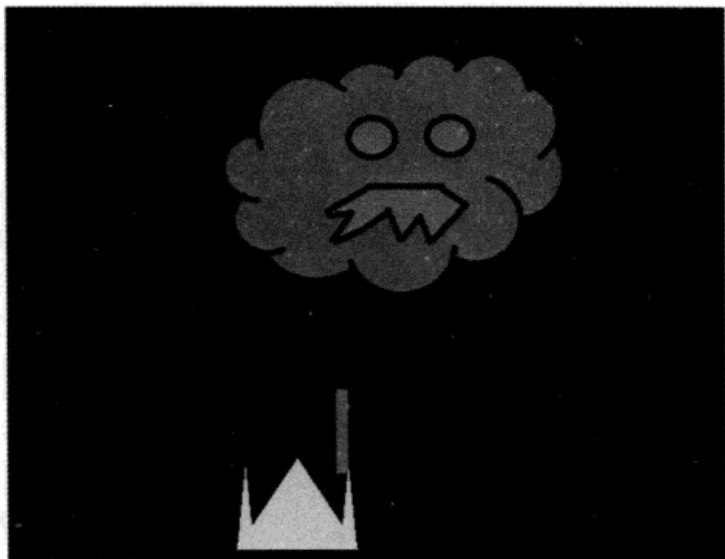


图 7-2

7.5 在 HTML5 的 canvas 元素中创建动画

Web 技术的最新发展，使在网站中创建令人惊叹的特效变得更加轻松。在 HTML5 中新增加了一个重要的元素：**canvas**。该元素的功能是在 HTML 中创建一个用于绘制位图的区域。**canvas** 标记的代码非常简单：

```
<canvas id="drawingSpace" width="300" height="200"></canvas>
```

要使 **canvas** 产生实际的用途，还需要 JavaScript 的帮助。下面的例子演示了如何在 **canvas** 上创建一个矩形：

```

$(function(){
    var rect_ctx = $("canvas#rect")[0].getContext("2d");
    rect_ctx.fillRect(10, 10, 50, 50);
});

```

在上面的代码中，首先选中了 **canvas** 对象，获取了 **canvas** 绘图平面的句柄(handle)。绘图平面用于绘制一个矩形，该矩形的起点为(10,10)，宽度和高度都是 50。绘图平面是一个对象，它包含了在 **canvas** 上绘图的所有绘制方法和属性。在上面的例子中，使用 **fillRect()** 方法创建了一个矩形，它具有系统默认的颜色：黑色。

canvas 具有自己的坐标系，它的原点在 **canvas** 绘图区域的左上角(0,0)，它 x 轴的值

沿水平向右增长，它的 y 轴则沿垂直方向向下增长。图 7-3 描绘了 canvas 的坐标系。

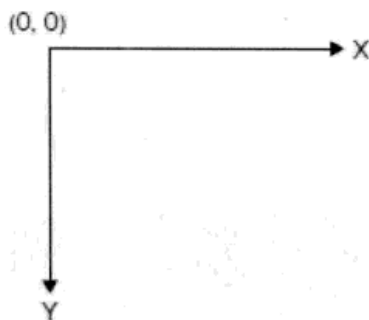


图 7-3

`strokeStyle` 就是图形的外轮廓样式，`fillStyle` 属性可用于绘制区域内的任何图形。下面的代码使用不同的 `fill` 和 `stroke` 属性绘制了两个不同的矩形：

```
$(function(){
    var ctx = $("#canvas#cl")[0].getContext("2d");

    ctx.fillStyle = "rgb(0,150,0)";
    ctx.strokeStyle = "rgb(200,0,0)";
    ctx.fillRect(0,0,150,150);

    ctx.fillStyle = "rgb(100,100,0)";
    ctx.fillRect(150,150,150,150);
});
```

此外还可以绘制连续的线段。只需要首先标记出绘图路径，然后再执行实际的绘制工作。

```
ctx.beginPath();
ctx.moveTo(1,1);
ctx.lineTo(10,10);
ctx.stroke();
ctx.closePath();
```

要想深入学习 canvas 元素，推荐参考 Mark Pilgrim 的《Dive Into HTML5》中关于 canvas 的章节。网站是 <http://diveintohtml5.info/canvas.html>。

在下面的例子中，重新编写了移动 div 方块的动画，但这次使用 canvas 元素代替了 div 元素。这个示例只是一个简单的黑色方块，但相同的原理可以应用于更加复杂的绘制对象。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <style type="text/css">
      canvas {
        width:100px;
        height:100px;
        top:0;
```



```

        left:0;
        position:absolute;
    }
</style>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script type="text/javascript">
    $(function(){
        var rect_ctx = $("#canvas#rect")[0].getContext("2d");
        rect_ctx.fillRect(0, 0, 50, 50);
        $(document).click(function(){
            $("#rect").animate({
                left : '+=800px'}, 200);
        });
    });
</script>
</head>
<body>
    <canvas id="rect"></canvas>
</body>
</html>

```

代码片段 canvas.txt

7.6 小结

本章介绍了如何使用 canvas 标记绘制图形、如何创建一个具有滑入动画效果的消息栏，还初步创建了一个具有一个“太空入侵”风格的小游戏。本章的前半部分详细地介绍了如何为 DOM 元素创建动画效果，可以将相同的技术应用于 canvas 绘制的元素。在第 8 章中，将继续探索如何使用 jQuery UI 来创建用户界面，jQuery UI 是 jQuery 核心库的一个扩展。

7.7 参考

<http://api.jquery.com/animate/>
https://developer.mozilla.org/en/canvas_tutorial
<http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#2dcontext>
<http://dev.opera.com/articles/view/html-5-canvas-the-basics/>

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 II 部分

jQuery应用

- 第 8 章：jQuery UI 第 I 部分——更轻松地创建 Web 界面
- 第 9 章：jQuery UI 第 II 部分——鼠标交互
- 第 10 章：编写高效的 jQuery 代码
- 第 11 章：jQuery 模板
- 第 12 章：编写 jQuery 插件
- 第 13 章：使用 jQuery Deferred 对象进行高级异步编程
- 第 14 章：使用 QUnit 进行单元测试

jQuery UI 第 I 部分——更轻松地 创建 Web 界面

本章内容

- jQuery 中的主题和样式
- 使用 ThemeRoller
- 使用 jQuery UI 小组件

jQuery UI 是隶属于 jQuery 的一个用户界面库，它包含了小组件、特效、动画和交互功能。jQuery UI 库中的小组件都是可主题化的，ThemeRoller 是一个 Web App，它可以简化创建主题的过程。使 jQuery UI 具有高级外观的效果。jQuery UI 框架还包含了一个非常完备的 CSS 类的集合，这些 CSS 类对于创建应用程序的主题非常有用。尽管其他插件也可能具有很多与之类似的功能，但 jQuery UI 具有统一的基础代码库和 API、可靠的总体质量、灵活的 UI 元素，这一切使得 jQuery UI 成为构建 Web 应用程序不可或缺的宝库。本章将介绍 jQuery UI 的主题、如何创建用户自定义主题、jQuery UI 小组件以及其他 jQuery 核心库未包含的特效。

8.1 主题和样式

使用 jQuery UI 非常简单。只需要先导航到 jqueryui.com/download，下载一个 jQuery UI 库的副本即可。既可以下载一个预定义的主题，也可以创建一个用户自定义的主题。如图 8-1 所示，在 jQuery UI 的下载页面中，还允许用户选择将哪些组件包含在下载库中。

几乎没有任何项目会使用到 jQuery UI 的所有小组件和工具，因此最好只下载必要的组件，以减小最终产品中需要下载的文件大小。对于浏览和学习而言，可以下载完整的 jQuery UI 库。

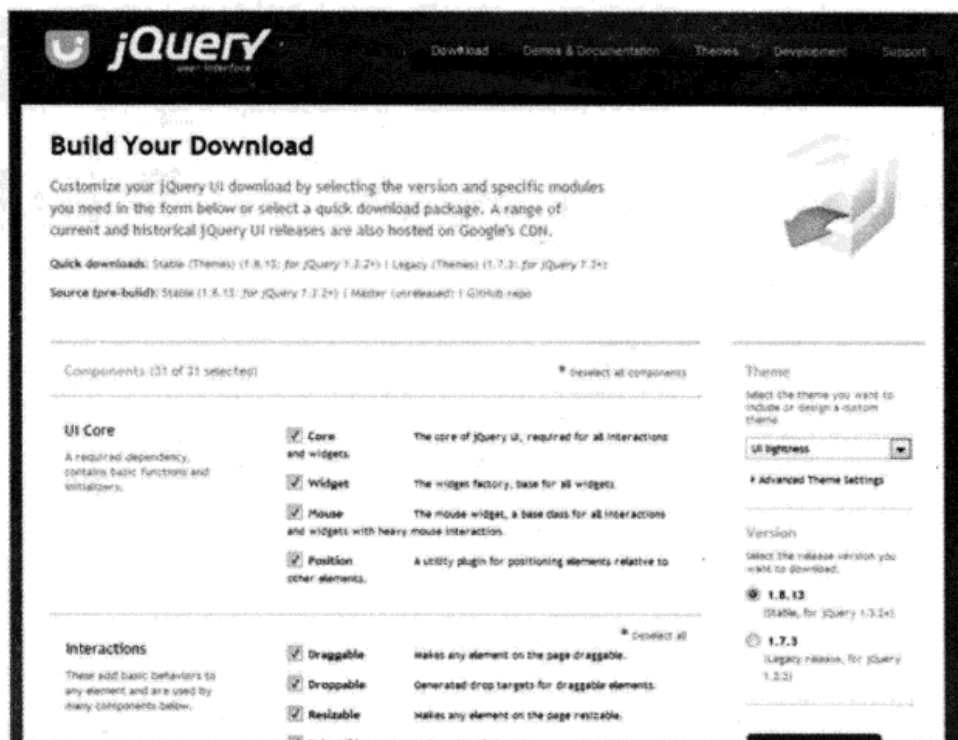


图 8-1

如果想编辑自己的主题，请打开 `css/selected-theme-name/jquery-ui-xxx.css` 文件，其中包含了一些注释行(在作者的文件中大约有 50 行)和一个 HTTP 地址。使用该地址可以在 ThemeRoller 中重新创建修改后的主题，并继续修改核心内容。

在下载 jQuery UI 库中包含了下列文件夹：

- **index.html**: jQuery UI 中包含的小组件和 CSS 类的一个概览。
- **js**: 该文件夹包含了最新的、完整版本的 jQuery UI 库的副本，还包含了一个 jQuery 核心库的一个副本。
- **css**: 该文件夹包含了一个默认的 jQuery UI 主题。
- **development-bundle**: 该文件夹包含了完整的、未精简版本的所有 jQuery UI 文件。

jQuery UI 包含了一个复杂而完备的 CSS 框架，它使所有小组件都具有一个统一的外观和效果。

CSS 框架的相关文件位于两个地方：一是在顶层的 `css` 文件夹中，二是在 `development-bundle/themes` 文件夹中。顶层 `css` 文件夹中的 `.css` 文件将所有 CSS 类的定义全部放在一个文件中，这可以减少获取 CSS 定义的 HTTP 请求的数量。位于 `development-bundle` 文件夹中的 CSS 文件，既包含了 `base` 主题，也包含了用户选中的主题。

8.2 使用 ThemeRoller

jQuery UI 是可主题化的(themeable, 这是 jQuery 维护者所使用的词汇), 也是用户可以自定义的。可以采用多种方法修改一个主题:

- 从头开始创建一个主题
- 使用一个预定义的主题, 不做任何修改
- 手工修改一个主题
- 使用 ThemeRoller 修改一个预定义的主题

如果从头开始创建一个主题, 必须自行创建大量的 CSS 类。使用默认主题是一个好办法, 但即插即用的默认主题可能并不符合应用程序的颜色要求, 因此最好的办法是对现有的主题进行修改, 以满足我们的需要。如果你愿意, 可以手工修改这些 CSS 类的定义, 但使用 ThemeRoller 可以免去不少麻烦。

可以从 <http://jqueryui.com/themeroller> 找到 ThemeRoller。在下载主题之前, 该页面显示了各个 jQuery UI 小组件和图标设置的一个预览。页面左侧是一个操作面板, 其中显示了 Roll your own、Gallery 和 Help 三个选项卡, 用户只需要选择一个看起来与最终产品需要的外观最接近的主题, 然后修改相应选项以满足需要即可。在完成修改之后, 单击 Download Theme 按钮就可以下载修改后的主题。这非常简单。

在任何情况下, jQuery UI 总是要求使用下面的代码模板, 这里列出了该模板以便于你使用:



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet"
      href="development-bundle/themes/base/ui.theme.css">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  </head>
  <body>
  </body>
</html>
```

代码片段 ui.css.txt

8.3 使用 jQuery 小组件

通常情况下, 用于创建小组件的 JavaScript 代码都非常简单:

```
$(selector).widgetType();
```

每一种类型的小组件都要求一种特殊的标记结构。例如, tabs 小组件要求的标记结构

是一个具有锚点的列表，它将这些锚点呈现为一个 tabs 小组件。表 8-1 列出了初始化每一种 jQuery UI 小组件所需的代码。

表 8-1 初始化 jQuery UI 小组件的代码

小 组 件	方 法 调 用
button	<code>\$('#myButton').button();</code>
button set	<code>\$('input[type=radio]').buttonset();</code>
tabs	<code>\$('#tabs').tabs();</code>
accordion	<code>\$('#myAccordion').accordion();</code>
autocomplete	<code>\$('#myText').autocomplete();</code>
datepicker	<code>\$('#date').datepicker();</code>
dialog	<code>\$('#myDialog').dialog();</code>
progressbar	<code>\$('#itemLoader').progressbar();</code>
slider	<code>\$('#numberSlider').slider();</code>

为了更好地控制小组件，小组件还可以接受一个包含多个配置选项的对象作为参数。

8.3.1 Button

默认情况下，浏览器使用 OS(操作系统)的原生按钮来呈现<input>按钮控件。这无疑是一种有效的选择，但开发人员常常希望创建自定义风格的按钮，使其与网站的整体设计相协调。从图 8-2 中可以看到，jQuery UI 的 button 类增强了原生的按钮。

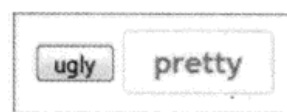


图 8-2

`$.button()`设置的样式应用于独立的 button，而`$.buttonset()`则应用于诸如单选按钮和复选框这样的按钮组。表 8-2 列出了每一种类型的 button 小组件可用的方法。

表 8-2 可用的\$.button()和\$.buttonset()方法

方 法	描 述
<code>\$(selector).button();</code> <code>\$(selector).buttonset();</code>	创建一个默认的 button 小组件
<code>\$(selector).button('disable');</code> <code>\$(selector).buttonset('disable');</code>	禁用之前创建的 button 小组件
<code>\$(selector).button('enable');</code> <code>\$(selector).buttonset('enable');</code>	启用之前创建的 button 小组件
<code>\$(selector).button('destroy');</code> <code>\$(selector).buttonset('destroy');</code>	彻底移除 button 小组件。该方法将使选中的元素恢复到初始状态

(续表)

方 法	描 述
<code>\$(selector).button('option', optionName, value);</code> <code>\$(selector).buttonset('option', optionName, value);</code>	获取或设置 button 的任意选项。如果省略了第三个参数, 则用于获取指定选项的值。否则, 则将把 optionName 设置为指定的值
<code>\$(selector).button('widget');</code> <code>\$(selector).buttonset('widget');</code>	返回.ui-button 元素
<code>\$(selector).button('refresh');</code> <code>\$(selector).buttonset('refresh');</code>	刷新 button 小组件的视觉状态

表 8-3 列出了 button 和 buttonset 可用的设置选项。

表 8-3 \$.button()和\$.buttonset()选项

选 项	接 受 值	描 述
Disabled	布尔值	禁用 button 小组件
Text	布尔值	选项, 指示是否显示文本
Icons	选项	在 button 小组件上显示的图标。可以包含一个主图标和一个次图标
Label	字符串	button 上的文本

从上面的选项列表中可以看到, 可以在 button 小组件上设置图标。一个 button 可以包含一个主图标(显示在 button 的左侧)和一个次图标(显示在 button 的右侧)。所设置的图标名称必须是有效的。例如:



可从
Wrox.com
下载源代码

```
$(selector).button({
    text : true,
    icons : {
        primary : 'ui-icon-alert',
        secondary : 'ui-icon-lightbulb'
    }
});
```

代码片段 ui-button-icon.txt

8.3.2 Tabs

选项卡(tabs)是一种组织信息的好办法, 它可以在单个页面中呈现密集的信息量, 并对内容按照分类进行组织, 一个选项卡就是一个 tabs 小组件, 它包含了多个选项页(tab)。要创建一个 tabs 小组件, 需要在标记代码中定义一个列表结构——既可以是一个有序列表, 也可以是一个无序列表, 并且每一个列表项中都包含有一个锚点元素。此外, 每一个列表

项必须包含一个对相应 tab 内容的引用。要创建一个简单的 tabs 小组件，首先应该创建出与下面代码示例类似的标记代码。在 tab 列表中，锚点元素的 href 属性引用了相应 tab 内容元素的 id。



可从
Wrox.com
下载源代码

```
<div id="tabs">
  <ul>
    <li><a href="#tab1">Tab #1</a></li>
    <li><a href="#tab2">Tab #2</a></li>
  </ul>
  <div id="tab1">Tab Number 1</div>
  <div id="tab2">Tab Number 2</div>
</div>
```

代码片段 *sample-tab-markup.txt*

表 8-4 列出了可用的\$.tab()方法。

表 8-4 可用的\$.tabs()方法

方 法	描 述
\$(selector).tabs();	在选中元素上初始化 tabs 小组件
\$(selector).tabs("destroy");	完全移除 tabs 小组件。该方法将使选中的元素恢复到初始状态
\$(selector).tabs("disable");	禁用 tabs 小组件
\$(selector).tabs("enable");	启用 tabs 小组件
\$(selector).tabs("option", optionName, [value]);	获取或设置 tabs 的任意选项。如果省略了第三个参数，则用于获取指定选项的值。否则，则将把 optionName 设置为指定的值
\$(selector).tabs("widget");	返回.ui-tabs 元素
\$(selector).tabs("add", url, label, [index]);	添加一个新的 tab 小组件。第 2 个参数 url 可以是当前页面内某个 tab 的标记代码片段的标识符，也可以是一个远程 tab(Ajax)的完整 URL。第 3 个参数 label 表示新 tab 的文本标签。第 4 个参数 index 是可选的，它表示一个基于 0 的位置，新 tab 将插入到该位置
\$(selector).tabs("remove", index);	移除一个 tab 小组件。第 2 个参数 index 指出要移除哪一个 tab
\$(selector).tabs("enable", index);	启用一个被禁用的 tab 小组件。第 2 个参数 index 指出要启用哪一个 tab
\$(selector).tabs("disable", index);	禁用 tab 小组件。请注意，不能禁用当前选中的 tab。第 2 个参数 index 指出要禁用哪一个 tab
\$(selector).tabs("select", index);	选中一个 tab 小组件，就像用鼠标点击了该 tab 一样。第 2 个参数 index 指出要选中哪一个 tab，或者是相应面板的 id 选择器
\$(selector).tabs("load", index);	重新加载一个 Ajax tab 的内容。第 2 个参数 index 指出要重新加载的是哪一个 tab

(续表)

方 法	描 述
<code>\$(selector).tabs("url", index, url);</code>	修改用于加载一个 Ajax tab 的 url。第 2 个参数 index 指出要移除哪一个 tab 的内容。第 3 个参数是要加载的新内容的 url
<code>\$(selector).tabs("length");</code>	获取匹配的第一个 tabs 小组件中所包含 tab 的数量
<code>\$(selector).tabs("abort");</code>	终止所有正在运行的 Ajax 请求和动画
<code>\$(selector).tabs("rotate", ms, [continuing]);</code>	建立一个自动流转的 tabs 小组件。第 2 个参数 ms 表示流转的时间间隔。第 3 个参数 continuing 用于控制在用户选中了一个 tab 之后, 选项页的流转是否继续执行

下面的例子演示了一个 tabs 小组件, 它包含了多个 tab, 它将选项卡的控制面板放在 tabs 小组件的底部, 并且可以重新排列。在初始化 tabs 小组件之后, 使用 `$.find()` 方法搜索 CSS 类为 `ui-tabs-nav` 的元素, 然后在匹配的元素集上调用 `$.sortable()` 方法。

`$.sortable()` 方法是 jQuery UI 的一个交互方法, 它允许使用鼠标拖动元素从而对元素进行重新排列。在第 9 章中将详细介绍 `$.sortable()` 方法。

下面的第二行代码调整了 CSS 类, 以允许 tabs 小组件将控制面板放在选项卡的底部。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
<link href="css/ui-lightness/jquery-ui-1.8.13.custom.css" rel="stylesheet" />
<script src="http://code.jquery.com/jquery-1.7.1.js"></script>
<script src="js/jquery-ui-1.8.13.custom.min.js"></script>
<script>
    $(function() {
        $("#tabs")
            .tabs()
            .find( ".ui-tabs-nav" )
            .sortable({ axis: "x" });
        $( ".tabs-bottom .ui-tabs-nav, .tabs-bottom .ui-tabs-nav > *" )
            .removeClass( "ui-corner-all ui-corner-top" )
            .addClass( "ui-corner-bottom" );
    });
</script>
<style type="text/css">
    #tabs {
        height: 200px;
    }
    .tabs-bottom {
        position: relative;
    }
    .tabs-bottom .ui-tabs-panel {
```

```
        height: 140px;
        overflow: auto;
    }
    .tabs-bottom .ui-tabs-nav {
        position: absolute !important;
        left: 0;
        bottom: 0;
        right: 0;
        padding: 0 0.2em 0.2em 0;
    }
    .tabs-bottom .ui-tabs-nav li {
        margin-top: -2px !important;
        margin-bottom: 1px !important;
        border-top: none;
        border-bottom-width: 1px;
    }
    .ui-tabs-selected {
        margin-top: -3px !important;
    }
</style>
</head>
<body>
    <div id="tabs" class="tabs-bottom">
        <ul>
            <li><a href="#t1">Tech Notes</a></li>
            <li><a href="#t2">Startup Ideas</a></li>
            <li><a href="#t3">JS Notes</a></li>
        </ul>
        <div id="t1">
            <p>Lorum Bacon</p>
        </div>
        <div id="t2">
            <p>Lorum Bacon</p>
        </div>
        <div id="t3">
            <p>Blah blah</p>
        </div>
    </div>
</body>
</html>
```

代码片段 sortabledtabs.txt

表 8-5 列出了\$.tabs()方法中一些可用的选项。完整的选项列表请参考 jQuery UI 的官方文档: <http://jqueryui.com/demos/tabs/>。

表 8-5 Tab 选项

选 项	接 受 的 值	描 述
disabled	布尔值	一个包含 tab 位置的数组, 当初始化 tabs 小组件时将禁用这些 tab
event	字符串	选中一个 tab 时触发的事件。通常允许在 mouseover 时选中一个 tab, 而不必按照执行一个标准的点击操作
fx	选项、数组	fx 允许设置一个动画, 用于显示或隐藏 tab 的内容
panelTemplate	字符串	如果使用的是 Ajax tabs, 可以使用 panelTemplate 来创建一个默认的 HTML 表单, 以便从中产生新的 tab 面板
selected	数值	一个基于 0 的整数, 用于设置初始选中的 tab
spinner	字符串	一个 HTML 字符串, 用于显示一个"spinner(提示)"或者其他信息, 以指示该 tab 正在加载。它要求在选项页标题的<a>标记中具有一个标记
tabTemplate	字符串	用于生成新 tab 的 HTML 模板

8.3.3 折叠面板(Accordion)

accordion 小组件与 tabs 小组件类似, 它是一种显示和组织大量信息的好办法。accordion 小组件是一个可展开、可折叠的面板, 它不会一次性就将所有内容展开显示, 而是每次只显示一个内容切片。

表 8-6 列出了可用的 accordion 小组件方法。

表 8-6 可用的\$.accordion()方法

方 法	描 述
<code>\$(selector).accordion();</code>	初始化一个 accordion 小组件
<code>\$(selector).accordion("destroy");</code>	完全移除 accordion 小组件。该方法将使选中的元素恢复到它的初始状态
<code>\$(selector).accordion("disable");</code>	禁用 accordion 小组件
<code>\$(selector).accordion("enable");</code>	启用 accordion 小组件
<code>\$(selector).accordion("option", optionName, [value]);</code>	获取或设置 accordion 小组件的任意选项。如果省略了第三个参数, 则用于获取指定选项的值。否则, 则将把 optionName 设置为指定的值
<code>\$(selector).accordion("widget");</code>	返回.ui-accordion 元素
<code>\$(selector).accordion("activate", index);</code>	激活 accordion 小组件的一个内容面板。index 是一个基于 0 的数值, 它是要激活内容面板的索引位置, 或者是一个与选择器匹配的面板元素的索引
<code>\$(selector).accordion("resize");</code>	重新计算 accordion 小组件内容的高度

要创建 accordion 小组件，要求标记代码具有这样的结构：一个 div 元素，其中嵌套了多个标题元素和 div 元素组，每一组表示 accordion 小组件中的一个折叠面板。在每个标题元素中要求包含一个链接。折叠面板的内容就放在标题元素之后的嵌套 div 元素中。比如下面的代码：



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <link href="css/ui-lightness/jquery-ui-1.8.13.custom.css" rel="stylesheet"/>
    <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
    <script src="js/jquery-ui-1.8.13.custom.min.js"></script>
    <script >
      $(function() {
        $( "#accordion" ).accordion();
      });
    </script>
  </head>
  <body>
    <div id="accordion">
      <h3><a href="#">First header</a></h3>
      <div>First content</div>
      <h3><a href="#">Second header</a></h3>
      <div>Second content</div>
    </div>
  </body>
</html>
```

代码片段 accordion.txt

表 8-7 列出了初始化 accordion 小组件时可用的一些选项。完整的选项列表，请参考 jQuery UI 官方文档：<http://jqueryui.com/demos/accordion/>。

表 8-7 accordion 选项

选 项	接 受 的 值	描 述
disabled	布尔值	禁用(true)或启用(false)accordion 小组件
animated	布尔值、字符串	设置首选的动画效果。如果参数是一个布尔值，则表示启用或禁用动画效果
event	字符串	一个表示事件类型的字符串，该事件用于触发 accordion 小组件的折叠或展开。通常允许在 mouseover 时改变选中的内容面板，而不必按照执行一个标准的点击操作

8.3.4 Autocomplete

由于 Web 上存在着大量的数据, 因此越快找到所需内容就越好。autocomplete 小组件用于在用户输入字符时, 显示一个预先定义的选项列表。如果你使用过任何一个搜索引擎, 那么对 autocomplete 功能应该早有体验。图 8-3 显示了在 Google.com 中一个与 autocomplete 功能类似的搜索框。

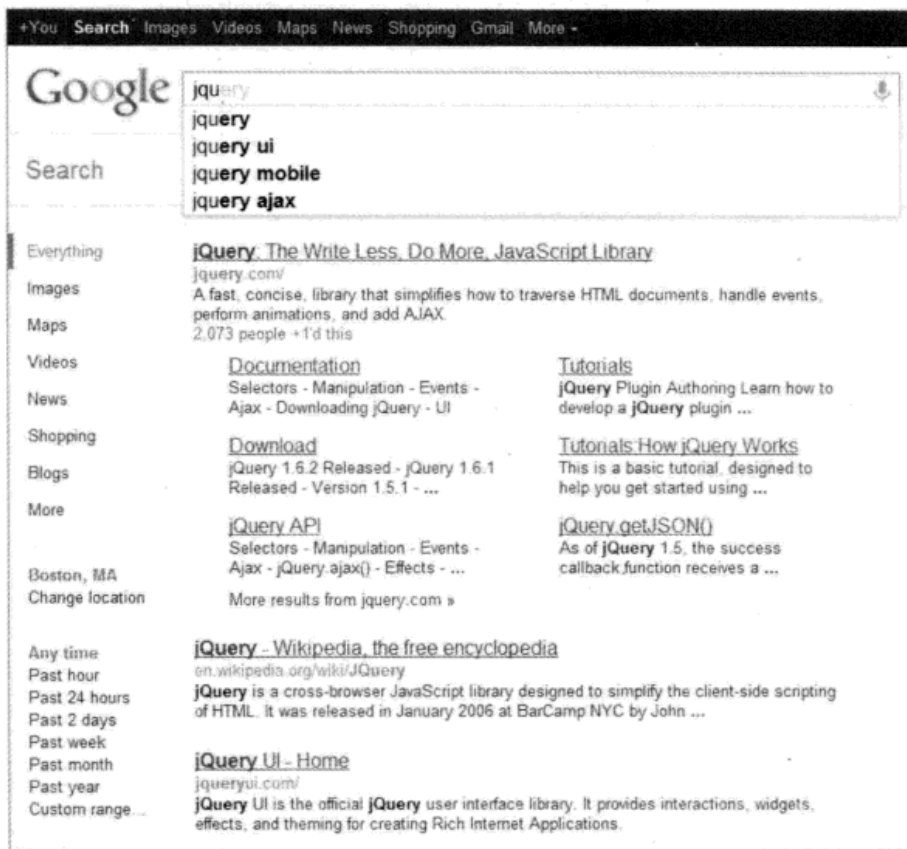


图 8-3

下面的代码示例简单地实现了一个 autocomplete 小组件。它使用了 source 选项以引用一个本地的数据源, 在本例中该数据源是一个数组。此外, 只需要一个简单的 \$.autocomplete() 方法调用, 就可以立即实现 autocomplete 的强大功能。要说明 jQuery UI 的简洁和强大, 或许这就是最好的例子。只需一行简单代码, 就可以显著增强网站或应用程序的可用性!



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
  <script src='http://code.jquery.com/jquery-1.7.1.js'></script>
  <link href="css/ui-lightness/jquery-ui-1.8.13.custom.css" rel="stylesheet"/>
  <script src="js/jquery-ui-1.8.13.custom.min.js"></script>
  <script>
    $(window).load(function() {
      $(function() {
        var stageWinners = [
          "Eddy Merckx", "Bernard Hinault",
```



```

        "André Leducq", "Lance Armstrong",
        "André Darrigade", "Mark Cavendish",
        "Nicolas Frantz", "Fran.ois Faber",
        "Jean Alavoine", "Jacques Anquetil",
        "René Le Greves", "Charles Pelissier",
        "Freddy Maertens", "Philippe Thys",
        "Louis Trousselier", "Gino Bartali",
        "Mario Cipollini", "Miguel Indurain",
        "Robbie McEwen", "Erik Zabel",
        "Jean Aerts", "Louison Bobet",
        "Raffaele Di Paco", "Maurice Archambaud",
        "Charly Gaul", "Walter Godefroot",
        "Gerrie Knetemann", "Antonin Magne",
        "Henri Pelissier", "Jan Raas",
        "Joop Zoetemelk", "Thor Hushovd"
    ];
    $( "#tdf" ).autocomplete({
        source: stageWinners
    });
});
});
</script>
</head>
<body>
    <div>
        <label for="tags">Tags: </label>
        <input id="tdf">
    </div>
</body>
</html>

```

代码片段 autocomplete.txt

表 8-8 列出了所有可用的\$.autocomplete()方法。

表 8-8 可用的\$.autocomplete 方法

方 法	描 述
\$(selector).autocomplete()	初始化一个 autocomplete 小组件
\$(selector).autocomplete("destroy")	完全移除 autocomplete 小组件。该方法将使选中的元素恢复到它的初始状态
\$(selector).autocomplete("disable")	禁用 autocomplete 小组件
\$(selector).autocomplete("enable")	启用 autocomplete 小组件
\$(selector).autocomplete("option", optionName , [value])	获取或设置 autocomplete 小组件的任意选项。如果省略了第三个参数，则用于获取指定选项的值。否则，则会将把 optionName 设置为指定的值

(续表)

方 法	描 述
<code>\$(selector).autocomplete("widget")</code>	返回.ui-autocomplete 元素
<code>\$(selector).autocomplete("search", [value])</code>	触发一个搜索事件。如果数据有效, 将显示一个自动完成的建议
<code>\$(selector).autocomplete("close")</code>	关闭 autocomplete 小组件菜单

表 8-9 列出了可用的 autocomplete 选项。

表 8-9 autocomplete 选项

选 项	接 受 的 值	描 述
disabled	布尔值	禁用(true)或启用(false)autocomplete 小组件
appendTo	选择器	要追加 autocomplete 列表的元素
autofocus	布尔值	指出第一个列表项是否应该自动获得焦点
delay	整数	以毫秒为单位的延迟, 在键盘击键后、autocomplete 列表打开之前的时延
minLength	整数	在触发 autocomplete 列表之前, 用户必须输入的最小字符数
position	对象	表示 autocomplete 小组件相对于关联的<input>元素的位置
source	字符串、数组、 回调函数	指定首选的数据源。数据源可以是一个包含本地数据的数组、一个表示 URL 的字符串、或者一个回调函数

8.3.5 Datepicker

datepicker(日历)小组件是一个功能更加丰富的 jQuery UI 小组件。如果你在网上预订过航班或酒店, 那么你一定使用过 JavaScript 的日历控件。

datepicker 小组件是即插即用的, 它极为有用, 大量的附加特性和选项可以扩展它的功能。除了可以采用很多办法来呈现 datepicker 之外, 它还包含了一组用于导航的工具函数和键盘快捷方式。可以采用下面的代码来调用默认的 datepicker 小组件:

```
$(selector).datepicker();
```

它将创建一个如图 8-4 所示的外观优美的日历。

这看起来很酷! 除了默认功能之外, 开发人员还可以对 datepicker 的功能进行大量的控制。表 8-10 列出了 \$.datepicker() 方法的完整列表。



图 8-4

表 8-10 可用的\$.datepicker()方法

方 法	描 述
<code>\$(selector).datepicker();</code>	初始化 datepicker 小组件
<code>\$(selector).datepicker("destroy");</code>	完全移除 datepicker 小组件。该方法将使选中的元素恢复到它的初始状态
<code>\$(selector).datepicker("disable");</code>	禁用 datepicker 小组件
<code>\$(selector).datepicker("enable");</code>	启用 datepicker 小组件
<code>\$(selector).datepicker("option", optionName , [value]);</code>	获取或设置 datepicker 小组件的任意选项。如果省略了第三个参数, 则用于获取指定选项的值。否则, 则把 optionName 设置为指定的值
<code>\$(selector).datepicker("widget");</code>	返回.ui-datepicker 元素
<code>\$(selector).datepicker("dialog", date , [onSelect] , [settings] , [pos]);</code>	以“对话框”方式打开一个 datepicker 小组件。该方法可以接收多个参数。 date: datepicker 小组件的初始日期 onSelect: 当选中一个日期时执行的回调函数 settings: datepicker 小组件的各种设置 pos: datepicker 的 top/left 位置, 即[x,y]坐标
<code>\$(selector).datepicker("isDisabled");</code>	判断 datepicker 小组件是否已经被禁用
<code>\$(selector).datepicker("hide");</code>	隐藏一个 datepicker 小组件
<code>\$(selector).datepicker("show");</code>	显示一个隐藏的 datepicker 小组件
<code>\$(selector).datepicker("refresh");</code>	刷新一个 datepicker 小组件
<code>\$(selector).datepicker("getDate");</code>	返回 datepicker 小组件中当前选中的日期。如果没有选中任何日期则返回 null
<code>\$(selector).datepicker("setDate", date);</code>	设置 datepicker 小组件中的当前日期

表 8-11 列出了 datepicker 小组件所有可用的选项。

表 8-11 datepicker 选项

选 项	接 受 的 值	描 述
disabled	布尔值	启用或禁用 datepicker 小组件的功能
altField	选 择 器 、 jQuery、 元素	定义一个从字段, 当主字段中的日期值更新时, 也将同步更新从字段。可以使用一个选择器或 jQuery 包装对象来指定一个从字段。常与 altFormat 选项配合使用

(续表)

选 项	接 受 的 值	描 述
altFormat	字符串	为从字段(altField)设置日期格式。它不必与主<input>字段的格式相匹配。关于格式化日期的完整选项, 请参考 \$.datepicker.formatDate()方法的文档: http://docs.jquery.com/UI/Datepicker/formatDate
appendText	字符串	在日期字段后追加的文本, 用于提示用户日期格式
autoSize	布尔值	日期字段是否应该缩放尺寸, 以适应 datepicker 当前格式的日期文本的长度
buttonImage	字符串	一个 URL 字符串, 用于设置弹出按钮的图片
buttonImageOnly	布尔值	当把该选项的值设置为 true 时, 将在<input>字段之后显示一个触发图片。当设置为 false 时, 图片将与一个 button 元素合并在一起
buttonText	字符串	设置触发按钮的文本内容
calculateWeek	函数	一个用户自定义的回调函数, 用于计算一个特定日期的属于哪一周
changeMonth	布尔值	添加一个选取月份的下拉列表
changeYear	字符串	添加一个选取年份的下拉列表
closeText	字符串	设置在 button 面板上 close 按钮上显示的文本内容。必须将 showButtonPanel 选项设置为 true
constrainInput	布尔值	如果设置为 true, 则允许使用在 dateFormat 中声明的日期格式对输入进行限制
currentText	字符串	在 button 面板上“当前日期”按钮所显示的文本内容
dateFormat	字符串	设置用于解析和显示日期的格式
dayNames	数组	一个长名称 day 的数组
dayNamesMin	数组	一个最短 day 名称的数组列表
dayNamesShort	数组	一个简短 day 名称的数组
defaultDate	日期、数值、字符串	设置 datepicker 的默认日期, 当打开 datepicker 时将高亮显示该日期。有效的输入选项包括: 一个日期对象、一个格式与 dateFormat 相匹配的输入字符串、距离当前日期的天数、一个包含多个值的字符串或者 null 值
Duration	字符串、数值	一个表示 datepicker 显示或消失速度的值
firstDay	数值	定义一周的第一天, 0 表示从星期天开始
gotoCurrent	布尔值	当设置为 true 时, “当前日期”按钮将链接到目前选中的日期, 而不是当日的日期

(续表)

选 项	接 受 的 值	描 述
hideIfNoPrevNext	布尔值	在“上一个月”、“下一个月”按钮不适用的情况下，隐藏这两个按钮，而不是禁用它们
isRTL	布尔值	如果当前语言是从右向左读的语言，则返回 true。这是一个重新引入的选项
maxDate	日期、数值、字符串	可选择的最大日期
minDate	日期、数值、字符串	可选择的最小日期
monthNames	数组	一个完整月份名称的数组，用于 dateFormat
monthNamesShort	数组	一个月份短名称的数组，用作 datepicker 的月份标题。
navigationAsDateFormat	布尔值	当设置为 true 时，在显示之前，\$.datepicker.formatDate() 函数将应用于 prevText、nextText 和 currentText 的值
nextText	字符串	在“下一个月”链接上显示的文本
numberOfMonths	数值、数组	声明在 datepicker 中显示几个月的日历，默认值为 1。它可以接受一个整数值，或者一个表示行数和列数的数组
prevText	字符串	在“上一个月”链接上显示的文本
selectOtherMonths	布尔值	当设置为 true 时，允许选择非当前月份的日期
shortYearCutoff	字符串、数值	设置截止年份，用于决定一个日期属于哪一个世纪
showAnim	字符串	一个用于设置显示或隐藏动画的选项
showButtonPanel	布尔值	显示或隐藏 button 面板
showCurrentAtPos	数值	当显示的月份多于一个月时，指定在哪一个月份中进行选择。它是一个基于 0 的索引，从左上角的月份开始计数
showMonthAfterYear	布尔值	在 datepicker 的标题中，年份是否显示在月份之前
showOn	字符串	有效的选项是 focus、button 或 both。它用于决定输入框获得焦点、单击鼠标按钮或者任意一种行为发生时，哪一种行为将触发 datepicker 小组件的显示
showOptions	选项	如果使用了 jQuery UI 特效来显示 datepicker，为 showAnim 设置附加选项
showOtherMonths	布尔值	是否在当前面板上显示上、下两个月的一些日期，这些日期是不可选的。要使这些日期可选，应该将 selectOtherMonths 设置为 true
showWeek	布尔值	显示星期数

(续表)

选 项	接 受 的 值	描 述
stepMonths	数值	单击“上一个月”、“下一个月”按钮后，一次翻几个月(跳过)
weekHeader	字符串	与 showWeek 配合使用，该字符串将作为星期列的列头标题。默认值为 Wk
yearRange	字符串	容许显示的年份范围
yearSuffix	字符串	在年份后显示的文本。例如：BC 或 AD

下面的例子演示了如何创建一个 2×2 的 datepicker，它还带有一个控制按钮：



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
  <link href="css/ui-lightness/jquery-ui-1.8.13.custom.css" rel="stylesheet"/>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script src="js/jquery-ui-1.8.13.custom.min.js">
  </script>
  <script>
    $(function(){
      $('#input#mainField').datepicker({
        appendText : '(mm/dd/yyyy)', dateFormat : 'dd/mm/yy',
        closeText : 'X',
        showOn : 'button'
        currentText : 'Now',
        numberOfMonths : [2,2],
        selectOtherMonths : true,
        showOtherMonths : true,
        showWeek : true,
        weekHeader : 'Week'
      });
    });
  </script>
</head>
<body>
  <input type="text" id="mainField">
</body>
</html>
```

代码片段 datepicker.txt

该示例的效果如图 8-5 所示。

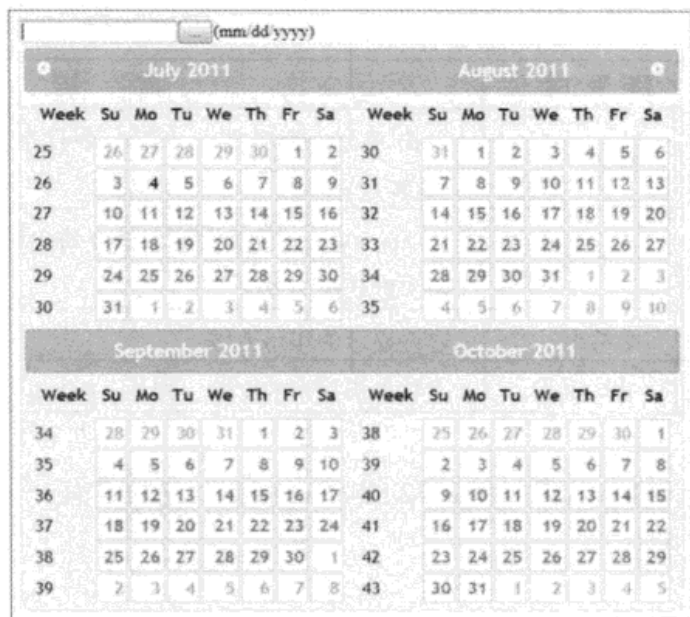


图 8-5

8.3.6 对话框

jQuery UI 对话框(dialog 小组件)要比 JavaScript 的 alert 消息框好用得多。

对于 JavaScript 原生的 alert 消息框, 一个较好的替代方案是使用 jQuery UI 的 dialog 小组件。比如下面的代码:



```
<!DOCTYPE html>
<html>
<head>
  <link href="css/ui-lightness/jquery-ui-1.8.13.custom.css" rel="stylesheet" />
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script src="js/jquery-ui-1.8.13.custom.min.js">
  </script>
  <script>
    $(function() {
      $( "#dialog" ).dialog({
        modal: true,
        buttons: {
          Ok: function() {
            $( this ).dialog( "close" );
          }
        }
      });
    });
  </script>
</head>
<body>
  <div id="dialog" title="A sample dialog">
    Message Goes here
  </div>
```

```
</body>
</html>
```

代码片段 *dialog.txt*

上面的代码将显示一个带有一条消息和一个 OK 按钮的对话框。OK 按钮的回调函数只是简单地关闭该对话框。请注意 `modal` 属性。在本例中将 `modal` 属性设置为 `true`，这表示这是一个模态对话框，在对话框打开期间，页面其余部分将不可访问。

表 8-12 列出了 `$.dialog()` 选项的完整列表。

表 8-12 可用的 `$.dialog()` 方法

方 法	描 述
<code>\$(selector).dialog()</code>	初始化 dialog 小组件
<code>\$(selector).dialog("destroy")</code>	完全移除 dialog 小组件。该方法将使选中的元素恢复到它的初始状态
<code>\$(selector).dialog("disable")</code>	禁用 dialog 小组件
<code>\$(selector).dialog("enable")</code>	启用 dialog 小组件
<code>\$(selector).dialog("option", optionName , [value])</code>	获取或设置 dialog 小组件的任意选项。如果省略了第三个参数，则用于获取指定选项的值。否则，则把 <code>optionName</code> 设置为指定的值
<code>\$(selector).dialog("widget")</code>	返回 <code>.ui-dialog</code> 元素
<code>\$(selector).dialog("close")</code>	关闭 dialog 小组件
<code>\$(selector).dialog("isOpen")</code>	如果 dialog 小组件当前是打开的，则返回 <code>true</code>
<code>\$(selector).dialog("moveToTop")</code>	在页面上具有多个 dialog 小组件的情况下，该方法将把选中的对话框移到 dialog 栈的顶端
<code>\$(selector).dialog("open")</code>	打开选中的 dialog 小组件

表 8-13 列出了 dialog 小组件所有可用的选项。

表 8-13 Dialog 选项

选 项	可接受的值	描 述
<code>disabled</code>	布尔值	禁用(<code>true</code>)或启用(<code>false</code>)dialog 小组件
<code>autOpen</code>	布尔值	如果将该选项设置为 <code>true</code> ，则 dialog 小组件将自动打开
<code>buttons</code>	对象、数组	定义在 dialog 小组件上要显示哪些按钮。它的值是一个单独的对象，该对象的属性是一个表示按钮类型的字符串，属性的值是一个回调函数，当单击该按钮时将调用该函数。此外，还可以传入一个“按钮/回调函数”对的数组，以定义要在 dialog 小组件中显示的一组按钮

(续表)

选 项	可接受的值	描 述
closeOnEscape	布尔值	定义当用户按下 <code>escape</code> 键时, <code>dialog</code> 小组件是否应该关闭
closeText	字符串	定义 <code>close</code> 按钮的文本
dialogClass	字符串	定义添加到 <code>dialog</code> 小组件的 CSS 类。以便更细致地设置 <code>dialog</code> 小组件的主题样式
draggable	布尔值	指示 <code>dialog</code> 小组件是否可以拖曳
height	数值	<code>dialog</code> 小组件的高度, 以像素为单位
hide	字符串、对象	当 <code>dialog</code> 小组件被隐藏时使用的效果
maxHeight	数值	<code>dialog</code> 小组件以像素为单位的最大高度
minHeight	数值	<code>dialog</code> 小组件以像素为单位的最小高度
maxWidth	数值	<code>dialog</code> 小组件以像素为单位的最大宽度
minWidth	数值	<code>dialog</code> 小组件以像素为单位的最小宽度
modal	布尔值	如果将该选项设置为 <code>true</code> , <code>dialog</code> 小组件将显示为一个模态对话框: 禁用与页面上其他元素的交互行为
position	字符串、数组	定义 <code>dialog</code> 小组件的位置
resizable	布尔值	指示 <code>dialog</code> 小组件的尺寸是否可以改变
show	字符串、对象	显示 <code>dialog</code> 小组件
stack	布尔值	指示 <code>dialog</code> 小组件是否放在 <code>dialog</code> 栈中其他 <code>dialog</code> 的顶部
title	字符串	<code>dialog</code> 小组件的标题
width	数值	<code>dialog</code> 小组件的宽度, 以像素为单位
zIndex	数值	<code>dialog</code> 小组件的初始的 <code>z-index</code> 值

8.4 进度条

用户需要知道某种操作的进度。比如用户想知道页面是否正在加载, 或者一个操作正在执行还是被卡住了。进度条就是一个经典的工具, 它可以将进度信息告知终端用户。值得注意的是, 进度条显示的进度值应该有一定的精确度, 否则显示的进度不准确, 可能会惹恼用户, 导致用户离开网站的页面。

与之前介绍的其他 jQuery UI 小组件相比, `progressbar` 小组件非常简单, 它只接受表 8-14 所示的两个选项设置。

表 8-14 Progressbar 选项

选 项	接 受 的 值	描 述
Disabled	布尔值	禁用(true)或启用(false)进度条
Value	数值	表示进度条的初始值

显然,要改变进度条的“完成百分比”,只需要改变 value 选项的设置。下面的例子演示了一个进度条,在页面加载之后,它缓慢地从 0%增加到 100%:



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
  <link href="css/ui-lightness/jquery-ui-1.8.13.custom.css" rel="stylesheet"/>
  <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
  <script src="js/jquery-ui-1.8.13.custom.min.js"></script>
  <script>
    $(function() {
      $( "#bar" ).progressbar({ value: 0 });
      setTimeout( updateProgress, 500 );
    });

    function updateProgress() {
      var progress;
      progress = $( "#bar" ).progressbar( "option", "value" );
      if (progress < 100) {
        $( "#bar" ).progressbar( "option", "value", progress + 1 );
        setTimeout( updateProgress, 500 );
      }
    }
  </script>
</head>
<body>
  <div id="bar">
  </div>
</body>
</html>
```

代码片段 progressbar.txt

8.5 滑动条

对于输入数值,滑动条更加具有约束能力,它的选项如表 8-15 所示。在使用文本框时,旧式的方法是使用掩码或验证对能输入的值进行限制,但这种方法存在一定的缺点。使用滑动条不但具有清晰的视觉效果,而且还非常直观。滑动条既可以是水平的,也可以是垂直的。


```

    });
</script>
</head>
<body>
    <div id="slider1"></div>
    <div id="slider2"></div>
    <div id="slider3"></div>
</body>
</html>

```

代码片段 slider.txt

该页面的输出如图 8-6 所示。

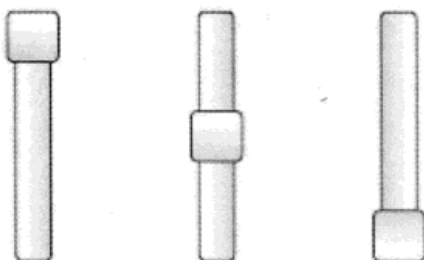


图 8-6

8.6 小结

本章简要地介绍了 jQuery UI 项目的结构，它包含了 jQuery UI 主题以及完整的 CSS 框架。另外，本章还介绍了一组精致的 jQuery UI 小组件，使用 jQuery UI 小组件有助于获得跨浏览器的一致的外观效果。每一个 jQuery UI 小组件都具有很多有用的选项。在第 9 章中，将进一步介绍 jQuery UI 与鼠标的交互功能。

8.7 参考

<http://jqueryui.com/docs/Theming>

<http://jqueryui.com/docs/Theming/API>

<http://jquery-ui.googlecode.com/svn/trunk/tests/static/icons.html>

<http://www.petefreitag.com/cheatsheets/jqueryui-icons/>

<http://docs.jquery.com/UI/Datepicker/formatDate>

http://www.ajaxlines.com/ajax/stuff/article/jquery_progressbar.php



Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 9 章

jQuery UI 第 II 部分——鼠标交互

本章内容

- 拖曳和置放元素
- 排序
- 缩放元素
- 使元素可选取

本章将继续介绍 jQuery UI 的特性，重点介绍使用鼠标对元素进行移动、排序、缩放和选取。与第 8 章中介绍的插件不同的是，它们不是 jQuery UI 小组件，而是添加到 DOM 元素的有趣行为。

默认情况下，div、span 等元素无法在 Web 页面上进行拖曳或缩放。需要在 JavaScript 的帮助下，才能使这些元素具有动态性。在 jQuery UI 库的支持下，要实现那些曾经在桌面环境中使用的操作方式，已经变得非常简单。

本章将介绍拖曳元素、置放元素、排序、缩放和使元素可选取。jQuery UI 已经为你处理了这些操作的绝大多数细节问题，并提供了一组选项集，以便使这些交互行为满足你的需要。

9.1 拖曳和置放

拖曳和置放元素通常一起使用。虽然有时只需要拖曳就够了，但通常需要同时实现拖曳和放置元素这两种功能。对于一个可拖曳的 DOM 元素，只需要在该元素上点击并按住鼠标，然后移动鼠标就可以在页面上拖动该元素。可置放元素指的是可以接收可拖曳元素的元素。

虽然拖曳和置放操作对于桌面型计算机是习以为常的功能，但在默认情况下，拖曳和

置放功能在 Web 页面并不可用。jQuery UI 使 Web 页面中的拖曳和置放功能变得简单和易于实现。尽管拖曳和置放是成对的操作，但它们要求分别进行调用。

通过调用 jQuery UI 的可拖曳方法 `$(selector).draggable()`，就可以使一个组件成为可拖曳元素，在 jQuery UI 中就是这样简单。与本书中已经介绍过的很多方法调用一样，`.draggable()` 方法是重载的，这减少了名称空间的杂乱。请参考表 9-1。

表 9-1 `$.draggable()` 方法

方 法	描 述
<code>\$(selector).draggable();</code>	使选中的元素变为可拖曳元素
<code>\$(selector).draggable("destroy");</code>	彻底移除可拖曳功能。该方法将使选中的元素恢复到它的初始状态
<code>\$(selector).draggable("disable");</code>	禁用可拖曳元素
<code>\$(selector).draggable("enable");</code>	启用可拖曳元素
<code>\$(selector).draggable("option", optionName, "value");</code>	获取或设置可拖曳元素的任意选项。如果省略了第三个参数，则用于获取指定选项的值。否则，则将把 <code>optionName</code> 设置为指定的值
<code>\$(selector).draggable("widget");</code>	返回 <code>.ui-draggable</code> 元素

表 9-2 显示了可拖曳元素的各种不同选项。

表 9-2 可拖拽功能的选项

选 项	接 受 的 值	描 述
<code>disabled</code>	布尔值	启用/禁用可拖曳功能
<code>addClasses</code>	布尔值	当设置为 <code>false</code> 时，阻止添加 <code>ui-draggable</code> 类
<code>appendTo</code>	元素、选择器	定义在拖曳期间， <code>helper</code> 元素附加到哪里
<code>axis</code>	字符串	限制拖曳仅能沿着 <code>x</code> 轴或 <code>y</code> 轴移动
<code>cancel</code>	选择器	阻止指定元素开始拖曳
<code>connectToSortable</code>	选择器	允许选中的可拖曳元素置放在一个可排序元素上
<code>containment</code>	选择器、元素、字符串、数组	将拖曳限制在一个指定的区域。有效的选项包括 <code>window</code> 、 <code>document</code> 、 <code>parent</code> 或一个 <code>[x1,y1,x2,y2]</code> 形式的数组
<code>cursor</code>	字符串	在拖曳期间使用的 CSS 光标类型，例如 <code>crosshair</code>
<code>cursorAt</code>	对象	调整拖曳的 <code>helper</code> 元素相对于鼠标指针的偏移位置
<code>delay</code>	整数	在拖曳生效之前，以毫秒为单位的延迟时间
<code>distance</code>	整数	在 <code>mousedown</code> 事件之后，鼠标必须移动的距离。只有超过此距离后拖曳才开始生效

(续表)

选 项	接 受 的 值	描 述
grid	数组	要求的值是一个形式为[x,y]的数组, 其中 x 和 y 分别定义了网格的间距像素。当声明了该选项之后, 拖曳的 helper 元素将与指定的网格对齐
handle	元素、选择器	定义一个拖曳“控制点(handle)”, 即限制从元素或区域的什么位置可以进行拖曳
helper	字符串、函数	允许在拖曳期间显示一个 helper 元素
iframeFix	布尔值、选择器	在拖曳期间, 阻止 iframe 捕获 mousemove 事件
opacity	浮点数	设置在拖曳期间, helper 元素的不透明度
refreshPositions	布尔值	当设置为 true 时, 在每次 mousemove 时, 都对可置放位置进行计算。这会降低应用程序的性能
revert	布尔值、字符串	如果设置为 true, 则当拖曳停止时, 可拖曳元素将返回它的起始位置
revertDuration	整数	回归到原始位置的整个动画过程的持续时间, 单位是毫秒
scope	字符串	用于分组可拖曳元素和可置放元素
scroll	布尔值	如果设置为 true, 则元素拖曳至边缘时, 父容器将自动滚动
scrollSensitivity	整数	一个以像素为单位的距离值, 表示当元素拖动至距离视口(viewport)边缘多少像素时, 应该滚动视口
scrollSpeed	整数	一旦鼠标进入到 scrollSensitivity 距离之内时, 视口滚动的速度
snap	布尔值、选择器	指示一个可拖曳元素是否应该与选中元素的边缘对齐
snapMode	字符串	定义对齐元素将附着的边缘。可能的值是 inner、outer 或者 both
snapTolerance	整数	定义一个以像素为单位的距离值, 它是距离对齐元素边缘的像素数, 如果小于此距离, 就应该执行对齐
stack	选择器	控制匹配元素的 z-index
zIndex	整数	被拖曳元素的 z-index

与.draggable()方法非常类似, \$.droppable()方法也是重载的。表 9-3 列出了所有可用的 \$.droppable()方法。

表 9-3 \$.draggable()方法

方 法	描 述
<code>\$(selector).droppable();</code>	使选中的元素成为可置放元素
<code>\$(selector).droppable("destroy");</code>	彻底移除元素的可置放功能。该方法将使选中的元素恢复到它的初始状态
<code>\$(selector).droppable("disable");</code>	禁用可置放元素
<code>\$(selector).droppable("enable")</code>	启用可拖曳元素
<code>\$(selector).droppable("option", optionName, "value");</code>	获取或设置可置放元素的任意选项。如果省略了第三个参数, 则用于获取指定选项的值。否则, 则把 <code>optionName</code> 设置为指定的值
<code>\$(selector).droppable("widget");</code>	返回.ui-droppable 元素

所有可拖曳元素都具有一个 `ui-draggable` 类。当处于正在拖曳的状态时, 该类变为 `ui-draggable-dragging` 类。

下面的例子同时使用了拖曳和置放两种交互效果, 演示了一种以可视方式划分任务的办法。在顶层是 3 个方块元素, 每一个方块代表了一个任务。在底层是任务的不同状态: `In Progress` 或者 `Finished`。通过 jQuery UI, 使任务方块成为可拖曳元素, 终端用户可以直观地对任务进行组织。一旦在表示 `In Progress` 状态的 `div` 元素中置放了一个任务, 该 `div` 元素的背景色就会改变。图 9-1 显示了该代码的输出效果。

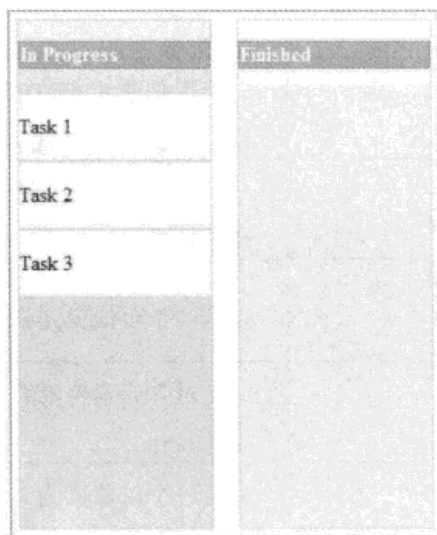


图 9-1



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
  <head>
    <link type="text/css"
      href="css/ui-lightness/jquery-ui-1.8.13.custom.css" rel="stylesheet"/>
    <script src="http://code.jquery.com/jquery-1.7.1.js"></script>
    <script src="js/jquery-ui-1.8.13.custom.min.js"></script>
```

```

<style type="text/css">
  div {
    width : 150px;
    margin-bottom : 20px;
    margin-right : 20px;
    float: left;
  }
  #flow {
    float: left;
    height : 400px;
  }
  #task3 {
    clear: both;
  }
</style>
<script type="text/javascript">
  $(function(){
    $('#div[id^="task"]').draggable({
      snap : 'div',
      snapMode : 'both'
      //snap tolerance
    });
    $('#flow').droppable({
      drop : function( event, ui ) {
        $( this ).addClass( "ui-state-highlight" );
      }
    });
  });
</script>
</head>
<body>
  <div id="task1" class="ui-widget-content">
    <p>Task 1</p>
  </div>
  <div id="task2" class="ui-widget-content">
    <p>Task 2</p>
  </div>
  <div id="task3" class="ui-widget-content">
    <p>Task 3</p>
  </div>
  <div id="flow" class="ui-widget-content">
    <p class="ui-widget-header">In Progress</p>
  </div>
  <div id="flow" class="ui-widget-content">
    <p class="ui-widget-header">Finished</p>
  </div>
</body>
</html>

```

代码片段 drag-drop.txt

9.2 排序

一组可排序元素就是一些可以重新排列顺序的可拖曳/置放组件：例如，一个列表，一系列元素，以及一组相互关联的元素。对于可排序元素，表 9-4 列出了各种可用的方法。

表 9-4 \$.sortable()方法

方 法	描 述
<code>\$(selector).sortable();</code>	使选中的元素变为可排序元素
<code>\$(selector).sortable("destroy");</code>	彻底移除可排序功能。该方法将使选中的元素恢复到它的初始状态
<code>\$(selector).sortable("disable");</code>	禁用可排序元素
<code>\$(selector).sortable("enable");</code>	启用可排序元素
<code>\$(selector).sortable("option", optionName, "value");</code>	获取或设置可排序元素的任意选项。如果省略了第三个参数，则用于获取指定选项的值。否则，则把 optionName 设置为指定的值
<code>\$(selector).sortable("widget");</code>	返回.ui-sortable 元素
<code>\$(selector).sortable("serialize",[options]);</code>	将可排序元素转换为一个适于提交的字符串
<code>\$(selector).sortable("toArray");</code>	将可排序元素转换为一个数组
<code>\$(selector).sortable("refresh");</code>	刷新可排序元素
<code>\$(selector).sortable("refreshPositions");</code>	刷新可排序项的缓存位置
<code>\$(selector).sortable("cancel");</code>	取消当前可排序元素的一次改变

在调用不同的方法时，还有很多选项可供配置，如表 9-5 所示。通过这些选项可以控制可排序元素的对拖曳和置放发生反应的敏感度和容忍度。可排序元素还可以连接两个不同的元素集。

可排序元素构建在拖曳和置放功能之上，但它具有一些特殊的行为。

表 9-5 可排序功能的选项

选 项	接 受 的 值	描 述
disabled	布尔值	启用或禁用可排序元素
appendTo	字符串	定义在拖曳期间， helper 元素附加到哪里
axis	字符串	限制拖曳仅能沿着 x 轴或 y 轴移动
cancel	选择器	阻止在特定元素上的排序
connectWith	选择器	将可排序元素与匹配指定选择器的另外一个可排序元素连接

(续表)

选 项	接 受 的 值	描 述
containment	元素、字符串、选择器	将拖曳限制在指定元素的边界内
cursor	字符串	定义排序时显示的光标
cursorAt	对象	调整拖曳的 helper 元素相对于鼠标指针的偏移位置
delay	整数	在拖曳生效之前, 以毫秒为单位的延迟时间
distance	整数	在 mousedown 事件后, 鼠标必须移动的距离。只有超过此距离后拖曳才开始生效
dropOnEmpty	布尔值	如果设置为 false, 可排序元素中的项不允许置放到一个与之链接的、空的可排序元素
forceHelperSize	布尔值	如果设置为 true, 则强制 helper 具有一个指定的尺寸
forcePlaceholderSize	布尔值	如果设置为 true, 则强制 Placeholder 具有一个指定的尺寸
grid	数组	要求的值是一个形式为[x,y]的数组, 其中 x 和 y 分别定义了网格的间距像素。当声明了该选项之后, 拖曳的 helper 元素将与指定的网格对齐。
handle	选择器、元素	定义一个拖曳“控制点”, 即限制从元素或区域的什么位置可以进行拖曳
helper	字符串、函数	允许在拖曳期间显示一个 helper 元素
items	选择器	声明元素内的哪些项可以排序
opacity	浮点数	设置在拖曳期间, helper 元素的不透明度
placeholder	字符串	应用于空白位置的 CSS 类
revert	布尔值/整数	如果设置为 true, 则当拖曳停止时, 可拖曳元素将返回它的起始位置
scroll	布尔值	如果设置为 true, 则元素拖曳至边缘时, 父容器将自动滚动
scrollSensitivity	整数	一个以像素为单位的距离值, 表示当元素拖动至距离视口边缘多少像素时, 应该滚动视口
scrollSpeed	整数	一旦鼠标进入 scrollSensitivity 距离内时, 视口滚动的速度
tolerance	字符串	在拖曳期间重新排序行为的方式。可能的取值为 'intersect'、'pointer'。在某些设置中, 'pointer' 方式更加自然
zIndex	整数	被拖曳元素的 z-index

下面的例子重新编写了任务列表的示例。在新示例中，每一个独立的任务框都具有一个 portlet 风格的界面。当对其中一个任务框重新排序时，可排序元素的新位置将使用一个不同的样式突出显示。图 9-2 显示了任务列表正在重新排序的过程。

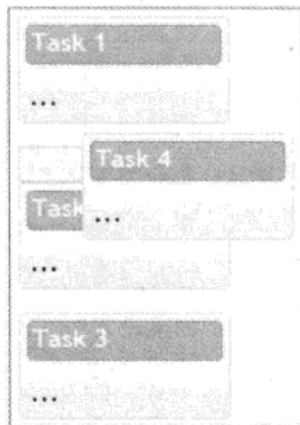


图 9-2



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
<link href="css/ui-lightness/jquery-ui-1.8.13.custom.css" rel="stylesheet" />
<script src="http://code.jquery.com/jquery-1.7.1.js"></script>
<script src="js/jquery-ui-1.8.13.custom.min.js"></script>
<style type="text/css">
    .column {
        width: 170px;
        float: left;
        padding-bottom: 100px;
    }
    .portlet {
        margin: 0 1em 1em 0;
    }
    .portlet-header {
        margin: 0.3em;
        padding-bottom: 4px;
        padding-left: 0.2em;
    }
    .portlet-header .ui-icon {
        float: right;
    }
    .portlet-content {
        padding: 0.4em;
    }
    .ui-state-highlight {
        height: 1.5em;
        line-height: 1.2em;
    }
</style>
```



```

<script>
$(function(){
    $( ".portlet" )
        .addClass("ui-widget ui-widget-content"+"ui-helper-clearfix
ui-corner-all")
        .find( ".portlet-header" )
        .addClass( "ui-widget-header ui-corner-all" )
        .prepend( "<span class='ui-icon ui-icon-minusthick'></span>" )
        .end()
        .find( ".portlet-content" );
    $('<div class='column'>').sortable({ placeholder: "ui-state-highlight" });
});
</script>
</head>
<body>
<div class="column">
    <div class="portlet">
        <div class="portlet-header">Task 1</div>
        <div class="portlet-content">...</div>
    </div>
    <div class="portlet">
        <div class="portlet-header">Task 2</div>
        <div class="portlet-content">...</div>
    </div>
    <div class="portlet">
        <div class="portlet-header">Task 3</div>
        <div class="portlet-content">...</div>
    </div>
    <div class="portlet">
        <div class="portlet-header">Task 4</div>
        <div class="portlet-content">...</div>
    </div>
</div>
</body>
</html>

```

代码片段 sortable.txt

9.3 缩放元素

当把缩放功能应用于一个元素时，将赋予元素一个控制点，并且可以使用鼠标可视化地缩放元素的尺寸。dialog 小组件就是一个可缩放小组件，在介绍 dialog 小组件时，我们已经间接地看到了缩放元素的功能。表 9-6 列出了所有可用的\$.resizable()方法。

表 9-6 \$.resizable()方法

方 法	描 述
<code>\$(selector).resizable();</code>	使选中的元素可缩放
<code>\$(selector).resizable("destroy");</code>	彻底移除元素的可缩放功能。该方法将使选中的元素恢复到它的初始状态
<code>\$(selector).resizable("disable");</code>	禁用可缩放元素
<code>\$(selector).resizable("enable")</code>	启用可缩放元素
<code>\$(selector).resizable("option", optionName, "value");</code>	获取或设置可缩放元素的任意选项。如果省略了第三个参数，则用于获取指定选项的值。否则，则将把 optionName 设置为指定的值
<code>\$(selector).resizable("widget");</code>	返回.ui-resizable 元素

由于缩放功能是一个非常基础的操作，表 9-7 列出了很多缩放功能的选项。与本章介绍的其他鼠标交互功能类似，默认的缩放功能可以满足很多需要，但它还提供了很多灵活的选项。

表 9-7 可缩放功能的选项

选 项	接 受 的 值	描 述
disabled	布尔值	启用/禁用可缩放功能
alsoResize	选择器、jQuery、元素	当缩放操作时，也缩放选中的元素
animate	布尔值	如果设置为 true，在缩放后以动画方式达到最终尺寸
animate Duration	整数、字符串	以毫秒为单位的动画持续的时长，其他可能的值是 slow、normal 和 fast
animateEasing	字符串	动画的缓动效果
aspectRatio	布尔值、浮点数	如果设置为 true，则按照元素的初始长宽比进行缩放
autoHide	布尔值	如果设置为 true，除非鼠标悬停于元素之上，否则隐藏选择的控制点
cancel	选择器	在特定元素上阻止缩放
containment	字符串、元素、选择器	将缩放限制在指定元素的边界内
delay	整数	在缩放起效之前，以毫秒为单位的延迟时间
distance	整数	在 mousedown 事件之后，鼠标必须移动的距离。只有超过此距离后 resize 事件才生效
ghost	布尔值	如果设置为 true，在缩放时将显示一个半透明的 helper 元素

(续表)

选 项	接 受 的 值	描 述
grid	数组	要求的值是一个形式为[x,y]的数组, 其中 x 和 y 分别定义了网格的间距像素。当声明了该选项之后, 拖曳的 helper 元素将与指定的网格对齐
handles	字符串、对象	如果将其设置为一个字符串, 则该字符串应该是一个以逗号分隔的下列可选值的列表: n、e、s、w、ne、se、sw、nw 或 all。 如果将其设置为一个对象, 可以使用与字符串相同的参数作为键。键值则是一个与可缩放元素的子元素相匹配的 jQuery 选择器, 它作为缩放操作的控制点
helper	字符串、函数	允许在缩放时显示一个 helper 元素
maxHeight	整数	可缩放元素允许缩放的最大高度
maxWidth	整数	可缩放元素允许缩放的最大宽度
minHeight	整数	可缩放元素允许缩放的最小高度
minWidth	整数	可缩放元素允许缩放的最小宽度

在下面的例子中, 创建了一个带有动画效果的可缩放小组件, 它的最小宽度和最小高度都是 300 像素、具有 ghost 效果和一个红色外边框的 helper 元素。填入的文本是使用 bacon ipsum 文本生成器生成的。Lorem ipsum 很好用, 但 bacon ipsum 更有趣。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
<link
  type="text/css"
  href="css/ui-lightness/jquery-ui-1.8.13.custom.css"
  rel="stylesheet" />
<script src="http://code.jquery.com/jquery-1.7.1.js"></script>
<script src="js/jquery-ui-1.8.13.custom.min.js"></script>
<style type="text/css">
  .ui-resizable-helper {
    border: 1px solid red;
  }
  #resize_me {
    border: 1px solid black;
    overflow : hidden;
  }
</style>
<script type="text/javascript">
  $(function(){
    $('#div#resize_me').resizable({
```



```

        animate: true,
        ghost : true,
        helper: "ui-resizable-helper",
        minWidth: 300,
        minHeight : 300
    });
});
</script>
</head>
<body>
    <div id="resize_me" class="ui-widget-content">
        <h2 class="ui-widget-header">Pork and Stuff</h2>
        Bacon ipsum dolor sit amet pastrami flank short ribs tongue, salami ham
        short loin shank Pancetta venison bacon shankle, swine jerky beef cow
        pork pork loin ham fatback beef rib salami ham hock. Salami beef bacon
        pork brisket, t-bone flank ball tip. Ham hock beef venison, t-bone
        andouille ribeye sirloin salami pork shankle. Ground round beef ribs
        tip, sirloin venison rump pork loin shoulder boudin salami flank chuck
        ham corned beef tenderloin. Tongue pork loin boudin, turkey ribeye salami
        pig biltong ham ham hock strip steak. Beef ribs short ribs turkey,
        pancetta swine pork meatloaf strip steak ham bacon corned beef short
        loin salami.
    </div>
</body>
</html>

```

代码片段 resizable.txt

这样的缩放效果是否平滑呢？图 9-3 显示了这个可缩放小组件的效果。

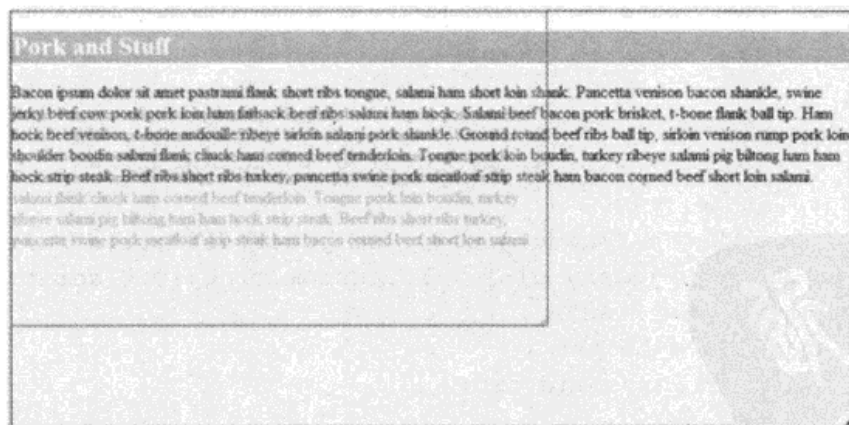


图 9-3

9.4 可选取元素

选取(selectable)是 jQuery UI 引入 Web 环境的另外一个桌面型操作范式。要选取一个可选取组件，只需要在一个空白区域上单击并按住鼠标，拖动鼠标到另外一个区域并释放

鼠标。或者只需要在组件内单击即可。

在选中的区域之外单击就可以取消选中的元素, 按住 Ctrl 再单击(或在 Apple 计算机上按住 Command 再单击), 可以选取多个元素项, 可选项功能的选项如表 9-8 所示。被选中的元素将被设置为 CSS 类 ui-selected。

表 9-8 可选项功能的选项

选 项	接 受 的 值	描 述
disabled	布尔值	启用/禁用可选项功能
autoRefresh	布尔值	在每一次选取操作开始时, 是否刷新每一个选中元素的位置和尺寸
cancel	选择器	在特定元素上阻止选取功能
delay	整数	在选取生效之前, 以毫秒为单位的延迟时间
distance	整数	以像素为单位的距离, 小于此距离时开始选取操作
filter	选择器	使匹配的子元素成为可选项元素

在下面的例子中, 结合了多个 div 元素, 所有列表项都是可选项的。



```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<link type="text/css" href="css/ui-lightness/jquery-ui-1.8.13.custom.css"
rel="stylesheet" />
<script src="http://code.jquery.com/jquery-1.7.1.js"></script>
<script src="js/jquery-ui-1.8.13.custom.min.js"></script>
<style type="text/css">
    #selectable div {
        width : 150px;
        height : 30px;
        padding : 10px;
        margin : 10px;
        border : 1px solid;
    }
    #selectable div.ui-selecting {
        background: blue;
    }
    #selectable div.ui-selected {
        background: lightblue;
    }
</style>
<script type="text/javascript">
    $(function() {
        $( "#selectable" ).selectable();
    });
</script>
</head>
```

```
<body>
  <div id="selectable">
    <div class="ui-widget-content unselectable">Item 1</div>
    <div class="ui-widget-content">Item 2</div>
    <div class="ui-widget-content">Item 3</div>
    <div class="ui-widget-content">Item 4</div>
    <div class="ui-widget-content">Item 5</div>
    <div class="ui-widget-content">Item 6</div>
    <div class="ui-widget-content">Item 7</div>
  </div>
</body>
</html>
```

代码片段 `selectable.txt`

对于上面例子中的可选项列表，图 9-4 显示了正在选取的状态和选取之后的状态。第一个 div 元素是不可选取的，因为它被赋予了一个 CSS 类 `unselectable`。

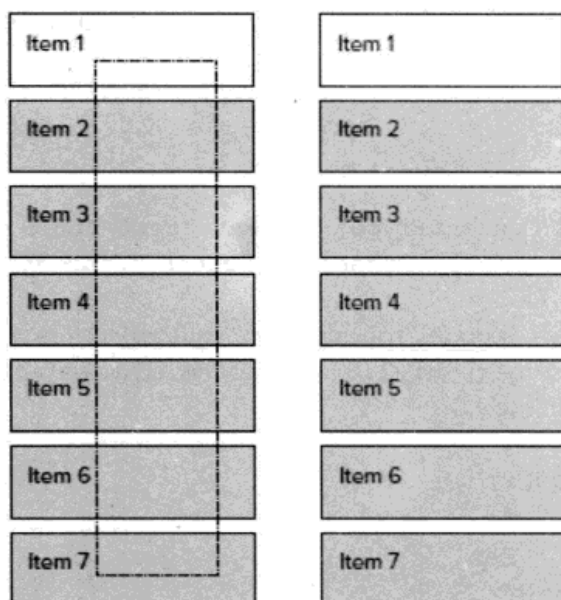


图 9-4

9.5 小结

本章完成了对 jQuery UI 的简要介绍。介绍了诸如拖曳和置放这样的交互性组件，以及如何在这些组件的基础之上创建更加复杂的交互性功能，比如重新排列元素。使用 jQuery UI 库，很容易创建自定义的 Windows 风格的小组件，并使之在 Web 上具有类似于 Windows 的操作功能。以前需要巨大工作量才能实现的功能，利用 jQuery UI 库只需要一个简单的方法调用就可以实现。对于每一种交互功能，本章还详细介绍了它的各种配置选项，这些选项说明，jQuery UI 是非常灵活的。

第 10 章

编写高效的 jQuery 代码

本章内容

- JavaScript 优化技术
- 使用 JavaScript 模式
- 使用 \$.data() 方法

经过前面几章的学习，我们已经掌握了 jQuery 所提供的功能。本章将介绍优化 jQuery 代码的方法和技术。

本章将介绍大量的 jQuery 优化技术、最佳实践和模式，可以立即将这些模式应用于代码之中，使代码更高效、更清晰，同时具有更好的可维护性。

这些优化技术来源于对 JavaScript 基础的深入理解、对 Web 浏览器中 DOM 与 JavaScript 引擎关系的深入理解，以及 Sizzle 和其他选择器引擎解析 CSS 选择器方法的深入理解。

本章还将介绍通用的 jQuery 模式，介绍如何使用这些模式，从而在各种类型和任何规模的项目中创建出清晰的、可维护的代码。

最后，本章还将介绍 jQuery 的 \$.data() 方法，开发人员可以利用 \$.data() 方法，以一种标准化的、清晰的方法存储和获取应用程序的数据。

在学习完本章的内容之后，你编写的代码将运行得更快，你的代码组织方式也将获得同事们的赞许。

10.1 优化技术

在这一小节中，将介绍一些简单的优化技术，在编写 jQuery 代码或者在编写纯粹的 JavaScript 代码时，应该将其铭记于心。

请注意，一旦开发人员开始严谨地关注性能问题，就会发现性能问题影响到所有代码。

高性能的代码将获取用户的青睐。

10.1.1 最小化 DOM 更新

优化前端代码最基础的概念之一，就是在任何交互活动中，保持 DOM 更新的数量绝对最小。从 JavaScript 引擎跨越到 DOM 是一种代价高昂的操作。当 DOM 更新导致重绘或重布局时尤其如此。

如果对重布局和重绘的概念不熟悉，请关注下一小节介绍。理解这些概念以及它们的差异、理解它们是如何影响 Web 页面或应用程序的感知和执行速度，是一个严谨的前端工程师必不可少的知识。

1. 重布局和重绘

重布局和重绘，是 Web 页面中最常见但也是最昂贵的两种操作。当浏览器需要更新它的呈现模型时(即将 DOM 与组成页面样式组件的 CSS 规则合并在一起)，就会发生重布局和重绘。只要加载页面，至少就会发生一次重布局和重绘(因为浏览器在加载页面时必须至少绘制一次页面)，但在动态页面的应用程序中，会发生很多次重布局和重绘。

重布局和重绘这两个事件是相关的，但又有细微的区别。理解二者的区别可以提高页面的性能。

- 当改变样式，而不改变整个页面的几何布局时，将发生重绘。隐藏一个元素，或者改变一个元素的背景色时，都将导致一次重绘。
- 当对页面的结构进行更新时，将导致重布局。从文档中移除元素，或者将元素添加到文档之中、改变元素的大小或改变元素可以导致重布局的属性，都会导致页面重布局发生。重布局的代价比重绘更大，因为它涉及使用当前 DOM/CSS 的定义，重新计算页面的几何结构。

2. 最小化 DOM 更新的实践

程序清单 10-1 是一个既简单又极端的例子，它说明了 DOM 更新对性能的损害。在第一个例子中，通过在每一轮 for 循环中调用 `$.append()` 方法，向一个 `<table>` 元素中追加了 10 000 个表格行。第二个例子则完全在 JavaScript 域中创建了一个包含 10 000 个表格行的 HTML 字符串，在循环结束之后，再一次将结果写入到 DOM 之中。



可从
Wrox.com
下载源代码

程序清单 10-1: 最小化 DOM 更新

```
for ( var i=0; i < 10000; i++ ){  
    $( "#main table" ).append( "<tr><td>My job is to log table rows, this is  
        row #" +i + "</tr></td>" );  
}  
  
var tableRows= "";  
for ( var i=0; i < 10000; i++ ){
```

```

    tableRows += "<tr><td>My job is to log table rows, this is row  

    #" + i + "</tr></td>";
}
$( "#main table" ).append( tableRows );

```

代码片段 *minimize-dom-updates.txt*

从表 10-1 中可以看到, 这种区别是显著的, 特别是在 Internet Explorer 8 浏览器中, 多次追加 DOM 的例子使用了将近 1 分钟才完成了操作。

表 10-1 最小化 DOM 操作的重要性

浏览器	10 000 次追加	1 次追加
Firefox 7.0	5 673 ms	421 5 ms
Chrome 15	9 372 ms	119 8 ms
Internet Explorer 8	50 783 ms	773 ms

运行 10 次的平均值

3. 充分利用 DOM Hyperspace

此外, 改进 DOM 操作的相关方法还包括: 在被称为 DOM 超空间(DOM Hyperspace)地方操纵 DOM。简而言之, 它指的是完全在 JavaScript 中创建或操纵 DOM 元素, 在操作完成之后, 仅一次性地将其更新到 DOM 之中。要实现这样的功能, 可以将已加载的 jQuery 节点创建为变量, 并用于操作; 或者使用 `$.detach()` 方法从文档的 DOM 中剥离要操作的元素。无论采用哪一种方法, 都是在操作完成之后, 再将 DOM 碎片插入到文档之中。程序清单 10-2 说明了这两种技术的基本原理。



可从
Wrox.com
下载源代码

程序清单 10-2: 在 JavaScript 中创建和维护元素

```

//使用 detach() 方法从文档中剥离元素
var $sideBar = $( "#sidebar" ).detach();
/*
生成侧边栏菜单、ads 已经其他元素
*/
$( "#main" ).append( $sideBar )

//使用 jQuery 创建要追加到文档的 DOM 元素
var $sideBar = $( "<aside id='sidebar'></aside>" ).detach();
/*
生成侧边栏菜单、ads 已经其他元素*/
//将其插入到文档的 DOM 之中
$( "#main" ).append( $sideBar )

```

代码片段 *dom-hyperspace.txt*

10.1.2 更高效的循环

在详细介绍这一优化技术之前，应该指出：在绝大多数情况下，在集合上使用 jQuery 便利的 `$.each()` 方法遍历数组的成员，这是一个可接受的、理想的解决方案。

但是在某些情况下，使用 `$.each()` 方法是不可接受的。对于巨大的集合，它的速度就会成为问题，此时使用传统的 JavaScript 控制结构将是一个更高效的办法。

1. `$.each()` 和 `Array.forEach()`：牺牲速度换取便利

无论是 jQuery 便利的 `$.each()` 方法，还是 EcmaScript 5 中的 `Array.forEach()` 方法，在功能的实现上都要比相应的 `for` 循环更慢。要理解这是为什么，首先需要理解循环代码运行的作用域。

2. 创建函数和增加符号查找层级的代价

让我们来看一下程序清单 10-3 中的例子，它遍历了一个整数的集合，检查其中的整数是不是 5 的倍数。第一种方式采用了传统的 `for` 循环，第二种方式采用 `[].forEach()` 方法，第三种方式则采用了 `$.each()` 方法。



可从
Wrox.com
下载源代码

程序清单 10-3: `$.each()`、`[].forEach()` 与传统 `for` 循环语句的对比

```
/*
 * 这是配套功能，生成一个整数数组
 */

var numbers = [],
    fives = [];
for ( var i=0; i<1000000; i++ ){
    numbers.push( Math.round( Math.random()*i ) );
}

/*
 * 第一种方法，使用传统的 for 循环
 * 将数组长度缓存在一个变量之中
 */
var test = numbers.length;
for ( var j = 0; j<test; j++ ){
    if ( ( numbers[j] % 5 ) === 0 && numbers[j] !== 0 ) {
        fives.push( numbers[j] );
    }
}

/*
 * 第二种方法，使用 ES5 的 forEach() 方法
 * 当这种方法可用时，jQuery 正是回退到使用该方法
 */
numbers.forEach(
```

```

function( e,I ){
    if ( ( e % 5 ) === 0 && e !== 0 ) {
        fives.push( e );
    }
}
)

/*
*最后, 使用 jQuery 便利的$.each() 方法
*/
$.each( numbers,
    function( i,e ){
        if ( ( e % 5 ) === 0 && e !== 0 ) {
            fives.push( e );
        }
    }
)

```

代码片段 looptest.txt

表 10-2 列出了这三种不同方法在速度上的差异。可以清楚地看到, for 循环明显要比另外两种方法更快。

表 10-2 for 循环、forEach()方法和\$.each()方法的对比(数值越高性能越好)

采用的方法	FF8	SAFARI 5	CHROME 15	IE9
For 循环	9 885	13 608	22 444	25 118
[].forEach	3 504	6 794	8 957	10 550
\$.each	1 981	6 794	3 609	4 268

<http://jsperf.com/book-loops>

理解二者之间的区别, 有助于开发人员编写出更高效的 JavaScript 代码。首先要提防额外的函数创建工作。因为它们接收函数作为参数, 这种便利性方法要求 JavaScript 引擎为每一轮循环创建一个新的执行上下文。而传统的循环语句在包含它的上下文中执行, 没有创建新执行上下文的负担。

作用域是第二个应该考虑的要素。因为所有操作都在函数中发生, 因此任何引用了外部作用域的变量都会被放在作用域链的上一层级中。循环体中的代码不得不反复地在作用域链中向上查找, 以检查引用的有效性。传统的循环则在一个作用域内执行, 在循环外定义的任何变量将依然保持在当前作用域中。

10.1.3 缓存对象

使用局部变量可以加快脚本的执行速度, 根据这一思路, 可以将对象缓存在局部作用

域中以减少脚本执行的额外负担。通常情况下，在函数中往往会多次使用到某个对象，此时将该对象缓存在一个局部变量中是非常有用的。这样可以缩短查找对象的时间，如果该变量是一个 jQuery 调用的结果，还可以减少 jQuery 根据选择器查找的次数。

在使用 `$(this)` 时常常需要这样做，因为绑定于一个函数的当前对象，常常是函数中多个操作所针对的目标对象。

作为一种编码风格，当保存到局部变量中的对象是一个 jQuery 对象时，在变量名之前添加一个 `$` 前缀是非常有用的。这可以向项目中的其他人员指明，该变量不是一个普通的纯变量，而是一个 jQuery 对象，它具有内建的 jQuery 方法。程序清单 10-4 给出了一个简单的例子，利用两个不同的变量说明了如何使用这一技术。



可从
Wrox.com
下载源代码

程序清单 10-4: 缓存对象以便重复使用

```
var $this = $( this ),
    $active = $( ".active" );
if ( $this.hasClass( "detail" ) ){
    if ( $this.hasClass( "next" ) ){
        $active
            .toggleClass( "hidden" )
            .removeClass( "active" )
            .next( ".details" )
            .toggleClass( "hidden" )
            .addClass( "active" );
    } else if ( $this.hasClass( "prev" ) ){
        $active
            .toggleClass( "hidden" )
            .removeClass( "active" )
            .prev( ".details" )
            .toggleClass( "hidden" )
            .addClass( "active" );
    }
}
```

代码片段 `caching-objects.txt`

10.1.4 高效使用选择器

一篇 10 年以前的文章能够在当今的 Web 开发社区中引起涟漪，这种情况非常少见。但对于 Mozilla 工程师 David Hyatt 写于 2000 年 4 月的一篇文章来说，情况的确如此。这篇文章名为 *Writing efficient CSS for use in the Mozilla UI* (https://developer.mozilla.org/en/Writing_Efficient_CSS)。它说明了浏览器解析 CSS 选择器的方式，指出了数种代价特别高昂的选择器模式。

这篇文章的要点包括：

- 解析引擎将自右向左计算每一条规则，它从关键(key)选择器开始，自右向左计算每一个选择器，直到发现一个匹配的选择器，如果没有找到匹配的选择器则放弃查找。
- 使用较底层的规则通常更有效率
- 尽可能使用具体化的选择器——ID 要比 tag 更好
- 避免不必要的冗余

最令人感兴趣的是 JavaScript 的选择器引擎，包括 Sizzle 引擎和内建的 document.querySelector，它们都使用了相同的技术，并且都对效率的共同关注中受益。

通常情况下，请保持选择器简单明了(比如充分使用 ID 选择器)，尽可能地使关键选择器更具体，无论对于 JavaScript 还是 CSS，这都可以加快网站的速度。

表 10-3 列出了 jQuery 中常用的选择器规则。可以清楚地看到，到目前为止，无论使用哪一种浏览器，使用 ID 选择器或单个类选择器都是选择元素最快的方式。在任何一种浏览器之中，其他更加复杂的选择器都要比 ID 选择器或类选择器慢很多。虽然并不总是能够使用 ID 或单个类来选择元素，但在可以使用 ID 选择器或类选择器的情况下，它们的速度是最快的。

表 10-3 jQuery 选择器的性能(值越高性能越好)

选 择 器	FF8	IE6	IE7	IE8	IE9
\$("#id")	177 115	7 497	47 530	52 402	294 060
\$(".class")	26 574	146	839	13 441	80 945
\$("section div div div #id")	8 933	976	7 343	13 441	20 291
\$("section div div div .class")	5 819	146	773	12 148	17 915
\$("section div div div p")	6 369	803	2 642	12 598	21 104
\$("section div div div p.class")	5 819	1 028	5 492	12 598	19 301
\$("section div div div p#id")	6 369	1 164	7 101	13 092	20 938
\$("section div div div p:not (.class)")	8 933	608	2 436	1 798	17 915

(续表)

选 择 器	ANDROID 2.3	SAFARI 5	FF 3.6	CHROME 15
<code>\$("#id")</code>	26 022	229 363	47 530	323 018
<code>\$(".class")</code>	6 051	66 117	3 683	63 522
<code>\$("section div div div #id")</code>	3 734	46 699	7 343	31 207
<code>\$("section div div div .class")</code>	1 565	21 535	2 734	12 148
<code>\$("section div div div p")</code>	1 633	21 104	2 734	13 652
<code>\$("section div div div p.class")</code>	1 633	23 087	2 734	14 666
<code>\$("section div div div p#id")</code>	3 734	46 262	2 866	23 646
<code>\$("section div div div p:not(.class)")</code>	1 565	19 981	2 866	12 042

<http://jsperf.com/book-selector-tests>

1. jQuery 特有选择器的性能

为了尽可能地提高效率,理解 jQuery 选择器的执行策略也是非常重要的。通常情况下, jQuery 尽可能地立即将选择器传递给一个原生的 DOM 方法。正如在上一小节中看到的那样,如果使用 `#id` 作为选择器, jQuery 将直接把该选择器传递给 `document.getElementById("id")` 方法。正如表 10-3 所示,该方法极为快速。出于同样的原因,可以传递给 `document.getElementsByTagName` 方法或 `document.getElementsByClassName` 方法的选择器都非常快速。

另外,如果你知道浏览器的特性列表,可能已经知道在低版本的 IE 浏览器中不支持 `document.getElementsByClassName` 方法,因此在 Internet Explorer 6 或 Internet Explorer 7 中,类选择器的速度较慢。

document.querySelector 和 jQuery 扩展

当 `document.querySelector(QSA)` 方法有效时,可以传递给 `document.querySelector` 方法的选择器也是非常快速的。请记住,对于 jQuery 中每一个有效的选择器,并不需要在 QSA 的原生实现中都有对应的可用实现。因为 jQuery 的选择器是一种 JavaScript 类库的实现,它不是针对浏览器优化的,因此 jQuery 对于 CSS3 选择器的扩展在速度上相对较慢。

表 10-4 列出了 jQuery 对于 CSS3 选择器的扩展。请谨慎地使用这些选择器。当使用这些选择器时,建议将一个纯 CSS 选择器传递给 jQuery,然后使用 `.filter()` 来筛选特定的结果。

表 10-4 jQuery 对 CSS3 选择器的扩展

选 择 器	选 取
:animated	选取在选择器执行时所有正处在动画过程中的元素
[name!="value"]	没有特定属性，或者具有特定属性但属性值不等于指定值的元素
:button	所有 button 元素
:checkbox	所有 checkbox 元素
:eq()	选取匹配元素集中索引为 n 的元素
:even	偶数元素
:file	所有 file 元素
:first	第一个匹配的元素
:gt()	选取匹配元素集中索引大于参数 index 的所有元素
:has()	选择至少含有一个与特定选择器匹配的元素
:header	所有 header 元素
:hidden	所有 hidden 元素
:image	所有 image 元素
:input	所有 input、textarea、select 和 button 元素
:last	最后一个匹配的元素
:lt()	选取匹配元素集中索引小于 n 的所有元素
:odd	奇数元素
:parent	选取所有是其他元素的父元素的元素
:password	所有 password 元素
:radio	所有 radio 元素
:reset	所有 reset 元素
:selected	所有 selected 元素
:submit	所有 submit 元素
:text	所有 text 元素
:visible	所有 visible 元素

<http://api.jquery.com/category/selectors/jquery-selector-extensions/>

2. 为选择器指定一个上下文

默认情况下，当把一个选择器传递给 jQuery 时，它将遍历整个 DOM。jQuery 方法还具有一个未充分利用的参数，即可以将一个上下文参数传入 jQuery，以限制它只搜索 DOM 中一个特定的部分。请注意，最好传入一个较快速的选择器作为附加的上下文。如果传入的上下文本身需要经过较慢的搜索才能达到，那么它对提升 jQuery 选择器的速度并没有多少帮助。

下面的代码将遍历整个 DOM:

```
$(".className");
```

而下面的代码仅搜索#id 元素:

```
$(".className", "#id")
```

如表 10-5 所示, 添加一个上下文总是比搜索整个 DOM 的选择器快速, 也比具有两个(或多个)选择器的组合选择器更快。

表 10-5 jQuery 选择器的性能(值越高越好)

选 择 器	FF8	IE6	IE7	IE8
<code>\$(".class")</code>	6 487	21	99	1 997
<code>\$("#id.class")</code>	2 595	1 376	8 457	2 697
<code>\$(".class", "#id")</code>	33 289	1 688	12 174	11 762

选 择 器	IE9	ANDROID 2.3	SAFARI 5	CHROME 15
<code>\$(".class")</code>	10 198	418	4 255	3 312
<code>\$("#id.class")</code>	3 313	420	5 100	5 199
<code>\$(".class", "#id")</code>	38 295	4 100	49 468	46 880

<http://jsperf.com/context>

10.1.5 考虑完全跳过 jQuery 方法

对于网站或应用程序来说, 如果性能是首先考虑的因素, 那么请记住在那些完全可以安全地跳过 jQuery 方法的地方, 回归到使用核心 JavaScript 方法和属性将更加快速。一个常见的例子, 就是使用 `attr()` 方法获取 `href` 属性, 然后再使用 `attr()` 方法将其应用于另外一个元素:

```
$("#baz").attr("href", $(this).attr("href"));
```

如果你了解核心 JavaScript 方法, 可以将上面这行代码改写为下面的代码:

```
document.getElementById("baz").href = this.href;
```

可以看到, 由于使用了 `document.getElementById` 方法来引用目标元素, 并直接查找这两个元素的 `href` 属性, 而非使用 `attr()` 方法, 因此消除了两次对 `attr()` 方法的调用。正如表 10-6 所示, 虽然这是一个简单的例子, 但在所有主流浏览器中, 使用原生方法的速度都更快。

表 10-6 使用原生 JavaScript 方法(值越高性能越好)

	FF8	IE6	IE7	IE8
原生方法	8 153	798	4 565	4 431
jQuery 方法	5 954	575	3 537	3 778

	IE9	SAFARI 5	CHROME 15
原生方法	7 512 1	4 905	11 796
jQuery 方法	6 532	12 166	10 801

<http://jsperf.com/sometimes-native-is-nice>

显然，由于存在跨浏览器问题，并非每一个 jQuery 语句都可以改为 JavaScript 原生方法。另外，使用原生方法将丧失 jQuery 链式方法调用的特性。如果各方面条件都允许，就不应该首先使用 jQuery 方法。上面的例子只是说明了在什么地方可以回归到使用原生方法，以便获得性能上的细微提升。常见的情形包括：简单地查找属性以获取一个字符串，以供后续操作使用或进行条件检测(比如 `this.id`、`this.className`)，你可以在自己的代码中找到类似的例子。如果性能是首要考虑的问题，那么从这些地方很容易获得性能的提升。

10.1.6 DRY

在软件开发领域，“请勿编写重复代码(Don't repeat yourself, DRY)”是一个反复强调的准则。对于可维护性和性能而言，这是一个重要的原则。对于简洁的 jQuery API，它也是一个重要的编码风格。在 jQuery 中，唯一鼓励重复性的地方就是自由地使用匿名函数。在 jQuery 中，将一个匿名函数传递给一个事件处理程序是如此地常见。在 jQuery 代码块中，出现多个执行相同功能的短小函数并不罕见。尽管短期内这样的代码感觉上更快速和更方便，但就长期而言简化重复代码是有益的。因为在后期维护代码时，没有人愿意费力地跨越数千行代码，在应用程序的多个地方重复地修改功能近似的代码。

程序清单 10-5 列出了一个典型的例子。该函数修改了所绑定元素的 CSS 类、触发一个 Ajax 请求，然后弹出一个 jQuery UI 对话框以显示该 Ajax 请求的结果。可以一次性将这些功能编写在一个匿名函数之中。这些功能都是简短和常见的代码。但如果这样的功能在整个网站中多次重复出现，就应该考虑进一步简化这样的代码。



可从
Wrox.com
下载源代码

程序清单 10-5: 重复的代码

```
$( "#foo" ).click(
    $( this ).addClass( "active" );
    $.get( "/my/app/service/", function( data ){
        $( "#dialog" ).text( data.status ).dialog( "open" );
    }
);
```

```

$( "#bar" ).click(
    $( this ).addClass( "active" );
    $.get( "/my/app/service/", function( data ){
        $( "#dialog" ).text( data.status ).dialog( "open" );
    }
);
$( "#baz" ).click(
    $( this ).addClass("active");
    $.get( "/my/app/service/", function( data ){
        $( "#dialog" ).text( data.status ).dialog( "open" );
    }
);

```

代码片段 repeating-yourself.txt

可以把多个选择器集合为一个选择器的列表，以简化上面的代码：



可从
Wrox.com
下载源代码

```

$( "#baz, #bar, #foo" ).click(
    $( this ).addClass( "active" );
    $.get( "/my/app/service/", function( data ){
        $( "#dialog" ).text( data.status ).dialog( "open" );
    }
);

```

代码片段 dry.txt

另外，如果在网站或应用程序各处都需要处理这样的事件，可以创建一个函数以供调用，这样就可以从任何地方访问该函数：



可从
Wrox.com
下载源代码

```

function ourHandler(){
    $( this ).addClass( "active" );
    $.get( "/my/app/service/", function( data ){
        $( "#dialog" ).text( data.status ).dialog( "open" )
    }
}
$( "#baz" ).click( ourHandler );

```

代码片段 dry-callable-function.txt

请对重复的代码保持警觉，一旦发现通常会产生重复代码的情况，就应该未雨绸缪，立即创建一个具有更好可维护性的方案。

10.2 使用 JavaScript 模式

软件设计模式的主题在深度上远远超过本书介绍的内容，理解和实现基本的设计模式，可以使开发人员在提高代码可维护性的道路上前进一大步。

要学习更多关于设计模式的内容，Erich Gamma、Richard Helm、Ralph Johnson 和 John Vlissides 四位作者的著作 *Design Patterns: Elements of Reusable Object-Oriented Software* 是必不可少的参考书籍。如果下面小节的内容引起了你的共鸣，请参考该书籍。

在下一小节中，将介绍控制全局名称空间污染、提供逻辑化的模块初始化以及使多个开发人员更容易协作的技术。

10.2.1 使用一个单例创建一个应用程序名称空间

用于改善代码结构最简单的技术之一，就是使用精心选择的名称空间。通常将这一模式称为一个单例(singleton)。

在介绍单例模式之前，先来看一下可用的非结构化组织代码的方法。这种风格的代码依然很常见，在 JavaScript 程序设计的早期曾经盛行一时。长期以来，这种方式只是简单地在全局空间中声明了一系列的函数定义。程序清单 10-6 列出了这种过时的代码风格：



可从
Wrox.com
下载源代码

程序清单 10-6：老式方法

```
function myAppInit() {
    //应用程序的初始化的代码
}

function myAppDashboard() {
    //仪表盘特性放在这里
}

function myControlPanel() {
    //控制面板代码放在这里
}

function myAppSettings() {
    //更新设置的代码放在这里
}

//通过 ready() 函数调用
$( document ).ready(myAppInit)
```

代码片段 multiple-functions.txt

从技术角度上来看，这种编写 JavaScript 代码的方式并没有任何错误，但它的确具有一些通用性上的障碍。

要查看它在通用性上的问题，只需要打开 Firebug 并选中 DOM 选项卡。你将看到所有这些函数都创建于全局名称空间下。如图 10-1 所示。

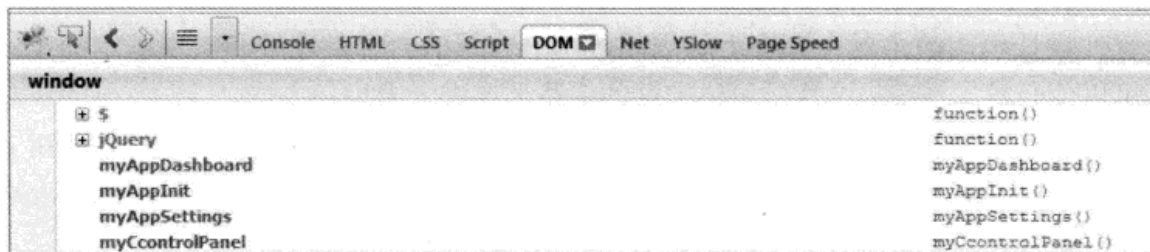


图 10-1

JavaScript 开发人员很快就会发现，在全局名称空间中创建很多变量很快就会导致严重的问题。比如一个显而易见的例子：在全局名称空间中定义了名为 `i`、`_` 或 `$` 的变量。这可能会导致后面的代码产生问题。

更复杂的命名或许会更加安全，但它们并非绝对安全。因此不要以为在一些无意义的字符之后命名变量就可以解决变量冲突的问题。

一旦项目中出现重复命名，就会产生变量名冲突的问题。

因此，除了将一系列的函数定义在全局名称空间之外，还可以充分利用单个变量来容纳应用程序的所有代码。最简单的例子就是将这样的变量定义为一个对象字面量，在该对象字面量中依次包含了应用程序的各个模块和属性。程序清单 10-7 列出了一个简单的例子。



程序清单 10-7：一个单例模式的例子

可从
Wrox.com
下载源代码

```
var myApp = {
  init : function() {
    //应用程序的初始化的代码
  },
  dashboard: function() {
    //仪表盘特性放在这里
  },
  controlPanel: function() {
    //控制面板代码放在这里
  },
  appSettings: function() {
    //更新设置的代码放在这里
  }
}
//通过 ready() 函数调用
$( document ).ready( myApp.init )
```

代码片段 singleton.txt

对于上面的新结构，打开 Firebug。可以看到仅有单个变量 `myApp` 出现在全局名称空间之中。如图 10-2 所示。

即便在这个简单的例子中，这也极大地降低了变量名冲突的可能性。如果应用程序具有数十个甚至数百个类似的方法或属性，应用程序名称空间的作用将更加显著。

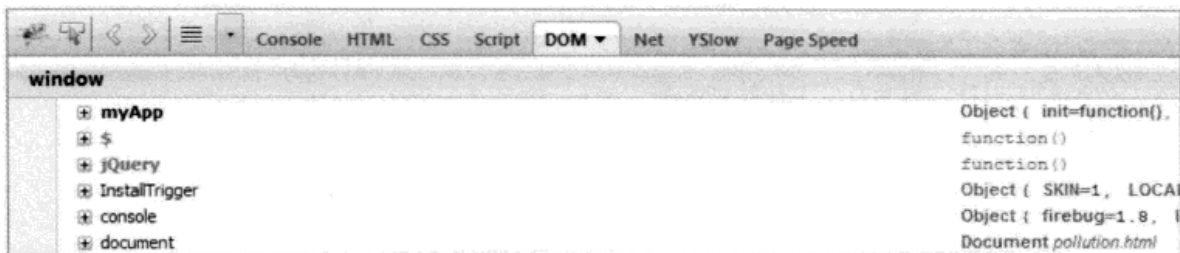


图 10-2

除了防止命名冲突的 bug 之外，名称空间的作用还远不止于此。可以对这个简单的单例进行扩展，为独立的应用程序特性或网站的部分创建单独的名称空间对象。

这一模式允许多个开发人员跨越多个文件，在单个应用程序名称空间之下进行工作。它具有更好的可维护性、减少了版本控制冲突的机会，或者说减少了开发人员相互踩到彼此脚趾的机会。虽然个别开发人员可能会侥幸逃脱命名冲突，但是在开发庞大 JavaScript 文件的团队中则可能导致灾难。

在程序清单 10-8 中可以看到，可以将之前定义的简单对象字面量拆分为多个应用程序对象。从代码结构的角度来说，可以将网站的每一部分拆分为一个独立的文件。比如本例中的 common.js、dashboard.js、controlPanel.js 和 settings.js 文件。除了 common.js 之外，每一个开发人员都有一个自己负责的网站部分和相应的 JavaScript 文件，开发人员只需要关注自己的 JavaScript 文件，完全不必担心是否会与其他开发人员的工作相冲突。common.js 文件由项目主管负责，一是为了便于维护，二是哪些内容是真正 common 的问题上，每个人的理解存在一定的差异。



可从
Wrox.com
下载源代码

程序清单 10-8：一个简单的应用程序框架

```
var myApp = {
  //Common 对象包含了应用程序各个部分都通用的代码
  common : {
    init : function(){
      //初始化应用程序的通用代码
    }
  }
};

myApp.dashboard = {
  //dashboard 特性也具有一个初始化函数
  init : function(){
    //初始化代码
  },
  update : function(){
    //更新 dashboard 的代码
  },
  render : function(){
```



```

        //呈现 dashboard 的代码
    }
};

myApp.controlPanel = {
    //控制面板特性也具有一个初始化函数
    init : function(){
        //初始化代码
    },
    settings : function(){
        //设置控制面板的代码
    }
};

myApp.settings = {
    //配置页面的初始化函数
    init : function(){
        //初始化代码
    },
    update : function(){
        //更新网站设置的代码
    }
}

```

代码片段 a-simple-application-framework.txt

请注意，尽管在开发时使用多个 JavaScript 文件是一个最佳实践，但实际上绝对不建议将多个 JavaScript 文件直接提供给用户。如果在项目中使用了多个 JavaScript 文件，应该采用某种办法自动地将这些 JavaScript 文件合并为单个文件，并且使文件体积最小化。要实现此类任务，一个较好的工具就是 HTML5 Boilerplate 项目及其相应的 build 脚本。可以在 <http://html5boilerplate.com/> 上找到该工具。另外，如果要静态 build 脚本，诸如 LABjs(<http://labjs.com/>)这样的脚本加载器提供了一种基于 JavaScript 的方法，可以根据需要加载相应的文件。

10.2.2 Module 模式

模块模式是单例模式的一个变种，它是 Douglas Crockford 创建的。Douglas Crockford 是 Yahoo! 公司首席 JavaScript 架构师，也是 JSON 数据交换格式的创建者。在 2007 年，Eric Miraglia 在 Yahoo! User Interface 的博客上介绍了这一模式(<http://www.yuiblog.com/blog/2007/06/12/module-pattern/>)，另外在 Crockford 的书籍 *JavaScript: The Good Parts* (O'Reilly 出版社 2008 年出版)中也深入介绍了模块模式。模块模式增强了单例模式提供的封装性，并增加了创建私有方法和私有属性的功能。

模块模式包含三个主要的组件：一个与前面例子类似的名称空间、一个立即执行的函数和函数的返回对象，该返回对象中包含了公有方法和公有属性。程序清单 10-9 给出了一

个简单的例子。



可从
Wrox.com
下载源代码

程序清单 10-9: JavaScript 模块模式

```
//app 的名称空间。传入 jQuery 对象以缩短查找过程
var myApp = function ( $ ) {
    //私有变量和方法，仅在该 myApp 中可用
    var message = "not directly accessible from outside the module";
    function multiplier ( x,y ) {
        return x * y
    };

    //返回对象包含了公有属性和公有方法
    return {
        init : function(){
            //初始化 app
        },
        prop : "42",
        specialNumber : function () {
            //访问私有方法
            var num = multiplier( 7 , 6 );
            return "Our special number is definitely " + num;
        },
        //对私有变量的提供有控制的访问
        shareMessage : function( arg ){
            if ( arg === "open sesame" ) {
                return message + ",unless you know the magic word";
            } else {
                throw new Error( "You need to know the magic word" );
            }
        }
    };
}( jQuery );
```

代码片段 module-pattern.txt

在控制台中测试该示例 app，可以看到该模块的工作原理。在模块内，可以引用私有变量，但在模块之外，无法直接调用私有变量。在返回的对象中可以通过方法来暴露私有变量，以便在特定情况下提供对私有变量的访问。程序清单 10-10 列出了在控制台中测试该示例模块的情况，它说明了以上两种情况。



可从
Wrox.com
下载源代码

程序清单 10-10: 在 JavaScript 控制台中测试模块模式

```
>>> console.log( myApp.message )
undefined

>>> console.log( myApp.multiplier() )
```

```
TypeError: myApp.privateMethod is not a function
```

```
>>> console.log( myApp.shareMessage( "please?" ) )
"You need to know the magic word"
```

```
>>> console.log( myApp.shareMessage( "open sesame" ) )
not directly accessible from outside the module, unless you know the magic word
```

```
>>> console.log( myApp.prop );
```

```
>>> console.log( myApp.specialNumber() );
Our special number is definitely 42
```

扩展该模式以增加更多的模块非常简单。程序清单 10-11 描述了一个 dashboard 模块，它包含了一个私有的配置对象和一个公有方法，该公有方法允许根据一组预定义的条件获得对 dashboard 配置的访问。如果你具有诸如 C#、Java 和 C++ 的编程经验，那么一定对 `private` 和 `public` 关键字的使用非常熟悉。JavaScript 中的模块模式提供了与之类似的功能。在 JavaScript 的开发中，并非时常都需要使用这种受保护的访问，但当我们需要实现这样的功能时，使用模块模式是非常重要的。



可从
Wrox.com
下载源代码

程序清单 10-11：一个 dashboard 模块

```
myApp.dashboard = function ( $ ) {
    //私有变量和方法
    var config = {
        "color" : "blue",
        "title" : "my dashboard",
        "width" : "960px"
    };
    return {
        init : function(){
            //初始化 dashboard
        },
        //updateConfig 方法允许更新私有 config 对象的配置
        updateConfig : function( obj ) {
            if ($.isArray(obj.color, ["red", "blue", "green", "purple"]) !== -1)
            {
                config.color = obj.color;
            }
            config.title = obj.title || config.title;
            config.width = obj.width || config.width;
        },
        render : function() {
            //呈现 dashboard
            var $dashboard = $( "<div/>" ).html( "<h1/>" );
            $dashboard.text( config.title )
                .css(
```



```

        { "width" : config.width,
          "color" : config.color }
      );
      $( "#main" ).append( $dashboard );
    }
  };
}( jQuery );

```

代码片段 *advanced-module.txt*

10.2.3 Garber-Irish 实现

下面将介绍 Garber-Irish 模式，它扩展了单例模式和模块模式，通过巧妙选择的标记和样式钩子(hook)，可以为网站和应用程序创建无缝的初始化模式。

Garber-Irish 模式是 Paul Irish 在一篇名为 Markup-based unobtrusive comprehensive DOM-ready execution 的博客文章中引入的([http://paulirish.com/2009/markupbased-](http://paulirish.com/2009/markupbased-unobtrusive-comprehensive-dom-ready-execution/)

[unobtrusive-comprehensive-dom-ready-execution/](http://paulirish.com/2009/markupbased-unobtrusive-comprehensive-dom-ready-execution/))，随后 Jason Garber 在他名为 *Extending Paul Irish's comprehensive DOM-ready execution*(<http://www.viget.com/inspire/extending-paul-irishs-comprehensive-dom-ready-execution/>)的博客中进行了修改。

简而言之，它充分利用了一个简短的工具脚本和 CSS 类、或者<body>元素上的 HTML5 的 data 属性，根据网站的区块或页面类型选择性地触发 init()事件。在一个应用程序中，多个\$(document).ready()方法调用常常分散在整个应用程序各处，这一模式缓解了捕获这些分散的\$(document).ready()方法调用的问题。它提供了一种统一的、结构化的办法，可以跨越多种范围工作(服务器端代码、JavaScript 和 CSS 样式)。

Garber-Irish 实现具有两种风格，开发人员可以根据自己的喜好选择使用其中一种版本。下面将分别介绍一下这两种实现风格。首先是 Paul 最初的版本。

1. 最初的方案

这一模式首先从定义一个典型的对象字面量开始。如程序清单 10-12 所示，这一解决方案具有一些特定的增强。请注意 init 方法和 common 小节，在 common 小节中包含了一个名为 finalize 的方法。Init 方法的设计目的是：当需要相关小节或特性时，在\$(document).ready()方法中调用 init 方法。这种需要取决于 HTML 的结构。finalize 方法用于包含需要尽快执行，但又不会影响到起始页面加载的代码。



可从
Wrox.com
下载源代码

程序清单 10-12: 对象字面量的增强，以支持 markup-based unobtrusive comprehensive DOM-ready execution

```

myApp = {
  common : {
    init : function(){

```

```

        //init()
    },
    finalize : function(){
        //稍后解释这里的代码
    }
},
dashboard : {
    init : function(){
        //dashboard 初始化
    },
    settings: function(){
        //dashboard 设置
    },
    render : function(){
        //呈现
    },
}
}
}

```

代码片段 *garber-irish-object-literal.txt*

该模式的第二个组件是一个简短的工具脚本，它的功能是解析<body>标记的 id 和 className 属性，并根据之前暴露的方法触发函数的执行。下面的代码直接取之于 Paul 最初的文章，程序清单 10-13 列出了这些代码。



可从
Wrox.com
下载源代码

程序清单 10-13: Markup-based unobtrusive comprehensive DOM-ready execution

```

/* from
 * http://paulirish.com/2009/markup-based-unobtrusive-comprehensive
 * -dom-ready-execution/
 * License: CC0 1.0 Universal (CC0 1.0) Public Domain Dedication
 */

UTIL = {

    fire : function(func,funcname, args){
        var namespace = myApp; //指定对象字面量的名称空间

        funcname = (funcname === undefined) ? 'init' : funcname;
        if (func !== '' && namespace[func]
            && typeof namespace[func][funcname] == 'function'){
            namespace[func][funcname](args);
        }
    },

    loadEvents : function(){

        var bodyId = document.body.id;

```

```

//首先调用 common
UTIL.fire('common');

//对所有其他类执行调用
$.each(document.body.className.split(/\s+/),function(i,classnm){
    UTIL.fire(classnm);
    UTIL.fire(classnm,bodyId);
});

UTIL.fire('common','finalize');

}

};

//在这里触发所有执行
$(document).ready(UTIL.loadEvents);

```

代码片段 irish.txt

最后一个组件是一个 HTML 文档，它的<body>标记中带有 class 属性和一个 id 属性，这两个属性分别对应于对象字面量中映射的网站 section 或特性。

```
<body id="settings" class="dashboard">
```

将上面的代码合在一起，将按照下面的顺序触发事件：

- myApp.common.init()
- myApp.dashboard.init()
- myapp.dashboard.settings()
- myapp.common.settings()

从上面的例子可以看到，可以对该模式进行扩展，使其不仅仅能应用于单个类。多个类可以联合起来，允许自动地触发多个特性。这一模式非常酷！因为一旦编写模式所需的组件，就可以初始化 JavaScript 特性，不必再编写任何一行 JavaScript 代码。使用 PHP 或 Java 的工程师只需要简单地在页面中追加一个类，模块中相应的初始化代码就会自动触发。

2. 使用 HTML5 的 data 属性代替 CSS 类和 ID

Jason Garber 模式是一个与 Paul 版本非常类似的变体，它的关键区别在于使用 HTML5 的 data 属性来触发事件的执行。

在 HTML5 中，data 属性是一种在 HTML 属性中存储用户数据的预定义方法。应用程序开发人员在 HTML 标记的用户自定义属性中存储字符串数据由来已久，因此在这种情况下，Web 超文本应用技术工作组(Web Hypertext Application Technology Working Group, WHATWG)不过是“新瓶装旧酒”，使用 data 属性来标准化开发人员常见的行为。标准化

`data` 属性的好处在于：`data-*`前缀支持以编程方式访问用户自定义的属性。在后面的小节中将更详细地介绍 `data` 属性，但就目前而言，在 `<body>` 标记中可以看到细微的区别：

```
<body data-controller="dashboard" data-action="render">
```

`Data` 属性取代了 `CSS` 类和 `ID` 属性，它可以将 `controller` 映射到 `action`。这种方法非常简洁，特别是在开发人员正在构建一个 `Ruby on Rails`(或类似)的应用程序时，其中统一的命名规范非常重要。另外如果需要的话，这种方法还允许将 `<body>` 标记的 `ID` 和 `class` 属性应用于其他的方法。

使用 `data` 属性的一个缺点是使特定区块的样式显得不太美观。在某种意义上，这种模式就是一种选择如何一一对应的方式——即与服务器上 `Ruby on Rails` 的 `controllers` 和 `action` 一一对应，或者与特定区块中的类和 `ID` 一一对应。

无论使用哪一种风格的 `Garber-Irish` 实现，这一模式提供了一个简洁的工具，为在 `$(document).ready()` 中运行代码提供了一个统一的、可伸缩和可维护的方法。

10.3 使用\$.DATA()

从前面的小节中可以看到，`HTML 5` 新的 `data` 属性被设计的目标，一是以结构化方式定义用户属性，二是以编程方式访问用户自定义属性。`jQuery` 提供了一个直接的方法 `$.data()`，用于创建或访问 `data` 属性。一旦在某个 `HTML` 元素上设置了任何 `data` 属性，就可以立即使用 `$.data()` 方法来获取 `data` 属性的值。`$.data()` 方法不但非常简便，它还提供了将任何数据绑定到一个 `DOM` 元素的功能。这提供了这样一种可能，可以像之前的字符串值一样，将数组和复杂的对象存储在用户自定义的 `data` 属性中。值得注意的是，虽然 `$.data()` 方法可以读取 `HTML5` 的 `data` 属性，但它实际上并不能以 `data-` 属性的格式向元素中写入值。这是合情合理的，因为并非任何数组和对象都适合存储为 `HTML` 属性中的字符串值。

这一功能非常强大，如果你之前曾实现过类似的功能，那么 `data` 属性正是你在开发工作中翘首以盼的功能。

如果你曾经将对象作为参数在多个方法之间传递，那么令人高兴的是往后再也不必这样做了——请拥抱 `$.data()` 方法吧。在多个方法调用中，使用跨越多个函数的参数链来管理应用程序状态的日子已经一去不复返了。现在，只需要简单地将应用程序或组件的数据直接绑定到 `DOM` 元素本身即可，这样一来，在应用程序的整个生命周期之内都可以直接访问这些数据。

另外，特别值得注意的是，`data` API 以一种避免循环引用的安全方式来存储数据，这避免了内存泄漏问题。

请注意对 `HTML 5` `data` 属性的支持情况：在 `jQuery 1.4.3` 中加入了对 `HTML5` `data` 属性的支持，并且在 `jQuery 1.6` 中发布了一个更加符合 `W3C` 规范的修正。

10.3.1 基本的.data() API

与其他很多 jQuery 方法类似, \$.data() 方法既可以作为一个属性设置器也可以作为一个属性获取器。如果向 \$.data() 方法传入两个参数作为一个“名称/值(name/value)”对, 或者传入一个“名称/值”对形式的对象, 那么 \$.data() 方法将根据参数设置数据的值。如果只向 \$.data() 方法传入一个字符串参数, 则 \$.data() 方法将返回该名称的 data 属性值。程序清单 10-14 列出了一些基本的例子, 字符串、布尔值、数组和对象都可以用作 \$.data() 方法的值。



可从
Wrox.com
下载源代码

程序清单 10-14: 简单的.data()示例

```
$('#dashboard').data('updated', 1321358806453);

$('body').data('fantastic', true);

$('#settings').data('faves', [1,4,6,8,9,14,27,42]);

$('#boston').data('bostonTeams', {
  'nba' : 'Celtics', 'nhl' : 'Bruins', 'mlb' : 'Red Sox', 'nfl' :
  'Patriots' });
```

代码片段 data-examples.txt

虽然 \$.data() 方法的 API 非常简单, 但通过上面的例子还不能说明该方法的强大功能和使用便利。\$.data() 方法以一种条理分明的、标准化的方式存储和获取应用程序的数据, 它清除了多年以来棘手的代码和雷同的模式。下面的小节将通过一个更加实用的例子说明 \$.data() 方法的优点。

在介绍下面的例子之前, 请注意虽然 \$.data() 方法允许存储页面级别的数据, 但它却根本无法存储持久化的数据。要想存储持久化的数据, 必须使用 cookie 或 HTML 5 新的 Storage API 进行处理, 这是浏览器中持久化数据的标准方式。

10.3.2 充分利用 Data API

程序清单 10-15 列出了一个应用程序的片段。在这段代码中, 初始化了一个小 Google Maps 应用程序。它创建了一个 Google 地图, 并使用 W3C 新的 Geolocation API, 根据用户的经度和纬度在地图上设置一个标记。

贯穿整个例子, 多次使用了 \$.data() 方法来存储变量、对象甚至 Google Maps 对象的实例。

在一开始, 存储了对 map 自身的一个引用。每一个 Google Maps API 都是从一个 Google Map 实例对象上调用的, 比如一个 map、geocoder 或者 marker。存储对 map 的引用可以简化后续的地图操作。另外还使用 \$.data() 方法存储了初始 mapOptions 对象。这是一种简单的

存储地图初始状态的方法，在任何需要的时候可以将地图恢复为它的初始状态。

随后，在 `success` 和 `failure` 这两个函数中，分别使用 `$.data()` 方法存储了一个 `geocoder` 对象和一个 `initialLocation` 对象。在后面的地址查找中，可以直接使用存储的 `geocoder` 对象，不必再进行初始化；根据地理位置查询所提供的新的经度和纬度，可以使用 `initialLocation` 对象的重置或重新居中地图。



可从
Wrox.com
下载源代码

程序清单 10-15: 在应用程序中使用 `.data()`

```
var loadMap = {
  init : function() {
    var GM = google.maps,
        defaultPosition = new GM.LatLng(42, -71),
        mapOptions = {
          zoom: 12,
          center: defaultPosition,
          mapTypeId: GM.MapTypeId.ROADMAP
        },
        map = new GM.Map( document.getElementById( 'map' ), mapOptions);

    $( "#map" ).data({ "map" : map, "mapOptions" : mapOptions });
    var success = function( data ){
      var position = new GM.LatLng(
        data.coords.latitude,
        data.coords.longitude ),
        niceAddress = "Your location",
        geocoder = new GM.Geocoder();
      $( "#map" ).data( "geocoder" , geocoder );
      geocoder.geocode(
        { 'latLng': position },
        function( results, status ) {
          if ( status == GM.GeocoderStatus.OK ) {
            if (results[0]) {
              niceAddress = results[0].formatted_address;
            }
          }
          var infowindow = new GM.InfoWindow({
            map: map,
            position: position,
            content: niceAddress
          });
          $( "#map" ).data( "infowindow" , infowindow );
        });
      map.setCenter( position );
      $( "#map" ).data( "initialLocation" , position );
    },
    failure = function( error ){
      var formResponse = function(e){
```



```

var geocoder = new GM.Geocoder(),
    position = defaultPosition,
    niceAddress = "Sorry We Couldn't Find Your Location";
$( "#map" ).data( "geocoder" , geocoder );
geocoder.geocode(
    { 'address': $("#location").val() },
    function( results, status ) {
        if ( status == GM.GeocoderStatus.OK ) {
            //设置位置
        }
        var options = {
            map: map,
            position: position,
            content: niceAddress
        },
        infowindow = new google.maps.InfoWindow(options);
        $( "#map" ).data( "infowindow" , infowindow );
        map.setCenter( options.position );
        $( "#map" ).data( "initialLocation" , position );
        $( "#geocode" ).hide();
    }
)
return false;
}
var $fallback = $( "<form id='geocode'></form>" );
if ( error ) {
    switch( error.code ) {
        /*错误处理*/
    }
}

fallback.append("<label for='location'>Enter Your Location"
    + "<input type='text' id='location' /></label>"
    + "<input type='submit' />");
fallback.bind("submit",formResponse);
$("#main").append( $fallback );
};
if (navigator.geolocation){
    navigator.geolocation.getCurrentPosition(
        success,
        failure,
        {timeout:5000});
} else {
    failure();
}
},
reset : function(){
    var map = $( "#map" ).data( "map" ),
        position = $( "#map" ).data( "initialLocation" ),

```

```
        infowindow = $( "#map" ).data( "infowindow" , infowindow );
        infowindow.close();
        map.setCenter( position );
    }
}
$( document ).ready( loadMap.init );
```

代码片段 data-api.txt

在这个简短的例子中，使用\$.data()方法以一种可访问的、标准化的方式存储了大量与应用程序有关的数据。在简单的 reset 方法中，只需要访问\$.data()方法存储的对象，就可以立即获得当前 map、infowindow 和创建地图时提供的经度和纬度等数据，这样一来很快就可以将地图重置为它的初始状态。一旦开发人员在应用程序中充分利用了\$.data()方法，就会为之前没有\$.data()方法时的艰苦工作感慨不已。

10.4 小结

本章介绍了大量改善 jQuery 和 JavaScript 代码的技术。通过本章的学习，你对常见的优化技术已经具有很好的理解，在这些优化技术的指引下，开发人员可以编写出更快、更高效的代码。从介绍浏览器更新 DOM 的方式开始，到理解 jQuery 便利性的代价和各种选择器的优化技术，你已经具有很好的技术储备，以优化和提高应用程序的运行速度。

另外，本章还介绍几个常用的 JavaScript 设计模式，采用这些设计模式可以创建出更统一、更具可维护性的代码。

最后，本章还简要介绍了 jQuery 的\$.data()方法，它允许开发人员以一种标准化的方式，简便而又安全地存储与 DOM 元素关联的应用程序数据。



第 11 章

jQuery 模板

本章内容

- 为什么要使用模板
- 模板的过去、现在和将来
- 创建和使用模板

本章将介绍 jQuery Template 插件。jQuery Template 是一种融合数据和标记代码片段的标准方法。如果你熟悉诸如 Freemarker 或者 Mustache 之类的模板引擎，将会对 jQuery Template 试图要实现的功能很熟悉。如果你已经做过基于 Ajax 的开发，那么必定创建过 HTML 代码块，用于根据 Ajax 请求的结果或用户的其他交互操作，将 HTML 代码块插入到文档中。典型情况下，这要求将字符串连接起来，或者使用原生的 DOM 方法创建元素，然后在将其插入到 DOM 中。jQuery Template 以标准化的方式简化了这种任务。

本章将介绍两种创建 jQuery 模板的方法：第一种方法是使用特殊的<script>标记来定义 jQuery 模板；第二种方法是使用纯粹的 JavaScript 通过 JavaScript 字符串来创建 jQuery 模板。本章将介绍如何使用有效的模板标记来更好地控制内容，还将介绍以最有效的方式来实现模板的策略。另外，还将介绍计划中的新版本 jQuery Template 插件的线路图。

11.1 征服字符串

如果你做过 JavaScript 开发，那么可能已经编写过类似于下面例子的代码。在这样的代码中，使用循环语句遍历了一个连环画图书集合中的数据，构造了一个 HTML 字符串，作为新的内容追加到文档中。



可从
Wrox.com
下载源代码

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<div id="main"> </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script src="//ajax.aspnetcdn.com/ajax/jquery.templates/beta1/
jquery.tmpl.min.js">
</script>
<script>
$(function(){
    var comics = [
        {
            imgSrc : "cover1.jpg",
            title : "Captain Templates",
            year : "2010",
            number : "1"
        },
        {
            imgSrc : "cover2.jpg",
            title : "Captain Templates",
            year : "2011",
            number : "2"
        },
        {
            imgSrc : "cover3.jpg",
            title : "Captain Templates",
            year : "2012",
            number : "3"
        }
    ]
    for( var i=0; i < comics.length; i++ ){
        $( "#main" ).append( '<div class="comic"><img src='
            + comics[i].imgSrc
            + '/><div class="details"><div class="title"><h3>'
            + comics[i].title
            + '</h3></div><div class="year">'
            + comics[i].year
            + '</div><div class="number">'
            + comics[i].number
            + '</div></div></div>'
        );
    }
});
</script>
```

```
</body>
</html>
```

代码片段 no-templates.txt

上面的代码定义了一个 `comics` 数组，遍历了数组成员，构造了 HTML 块并添加到文档中。该例子采用的方法是将 HTML 字符串与数据简单地连接起来。虽然这是一个通用的解决方案，但 jQuery 模板从多个重要方面改进了这一模式。

11.1.1 分离内容与行为

jQuery 模板可以进一步分离内容与行为。

现代 Web 开发的核心原则之一，就是结构、样式、行为的分离。即将描述页面内容结构的代码(HTML)，与定义样式(CSS)的代码和管理行为的代码(JavaScript)分离。从历史发展的角度看，这种分离首先是将诸如``等基于样式的 HTML 元素和诸如基于表格的布局模式，移到纯粹的 CSS 布局方式。其次，是将内联 CSS 和 JavaScript 从 HTML 文件中移除。这些都是积极的进步，但更加动态的网站和应用程序对开发技术提出了新的要求。越来越多的开发人员依赖于 JavaScript 来驱动网站，越来越多的样式、内容和结构都需要 JavaScript 的驱动才能正常工作。

当开发较小的项目时，JavaScript 驱动的方式是可以接受的，但是，一旦应用程序发展到数千或数万行代码，并且越来越多的开发人员希望能够管理这些代码时，这很快就会出现問題，诸如“这些标记代码是从哪里来的？”之类的问题将会越来越频繁地遇到。

jQuery 模板插件可以彻底分离文档结构和程序逻辑。这是编写可维护代码的关键。

11.1.2 代码重用

jQuery 模板提供了这样的功能：定义模板一次，就可以多次应用模板，包括与其他模板混合和匹配。使用 jQuery 模板，开发人员具有更多机会创建可重用的标记代码模式。

11.1.3 简洁而优美

一个好的模板系统应该简洁而又容易阅读。使用 jQuery 模板，开发人员不必把注意力放在正确地使用引号、加号和数据引用上，jQuery 模板可以创建清晰、统一的标记代码。

11.1.4 jQuery 模板的过去、现在和未来

在深入学习 jQuery 模板的细节之前，先了解一下 jQuery 模板的来历是有益的。

jQuery 模板是在 2010 年 3 月着手开发的，当 2010 年 5 月 John Resig 发布了一个 jQuery 模板的原型时，Boris Moore 正在开发一个类似的插件。他的工作分支最终合并到主流的 jQuery 官方插件中。随着对该插件的继续开发，最终达到了 beta 阶段。

在 2011 年 4 月，jQuery Template 插件的开发工作停止了，它永远地停留在了 beta 阶

段。在 jQuery 的官方博客上发表了一篇博客，宣布了这一改变(<http://blog.jquery.com/2011/04/16/official-plugins-a-change-in-the-roadmap/>)。它还宣布在 jQuery UI 项目的支持下，正在开始开发一个新的模板插件。

未来

原来 jQuery Template 插件的驱动力——Boris Moore——与 jQuery UI 团队一起，继续开发新的 jQuery 模板。在 2011 年 10 月，Moore 发表了一篇博客(<http://www.borismoore.com/2011/10/jquery-templates-and-jsviews-roadmap.html>)，更新了新插件工作进度的信息。在该博客中，他声明最终的解决方案将包含两个部分：

- JsRender 模板，被称为“下一代 jQuery 模板”，为高性能的、纯粹基于字符串的呈现做了优化，不依赖于 DOM 或 jQuery。
- JsViews 模板，它是“交互性的数据驱动视图，构建于 JsRender 模板之上”。

这个新的解决方案承诺改进速度、具有更高的灵活性、可以对表示和行为进行更好的分离。

现在

我们对这一新技术的发展充满信心，对于现代 Web 开发而言，一个稳健的模板系统是至关重要的。就目前而言，使用现有的模板插件是明智的，直到新的模板成熟到足以取代它。

11.1.5 创建 jQuery 模板

可以使用两种基本方法来创建 jQuery 模板。采用哪一种方法，取决于项目的特性、开发团队的组成和网站或应用程序的性能要求。

1. 创建内联模板

最简洁的办法，就是通过使用一个特殊的、精心设计的<script>标记来创建 jQuery 模板：



可从
Wrox.com
下载源代码

```
<script id="comics" type="text/x-jquery-tmpl">
  <div class="comic">
  <div class="details">
    <div class="title">
      <h3>{title}</h3>
    </div>
    <div class="year">
      {year}
    </div>
    <div class="number">
      {number}
    </div>
  </div>
</script>
```




```

    </div>
</script>

```

代码片段 `script-tag-template.txt`

该例子创建了一个名为 `comics` 的 jQuery 模板。在模板引擎中，该脚本的 `id` 将绑定于模板的名称。特殊的 `type` 属性向 jQuery 支持，该脚本是一个模板脚本块。浏览器本身将忽略该脚本块，因为浏览器并不知道如何处理 `text/x-jquery-tmpl` 类型的脚本。该示例还引入了第一个 jQuery 模板标记：`${}`，它向模板结构暴露变量。

在两种创建 jQuery 模板的方式中，这种方式更为清晰。它实现了结构与行为的完全分离。在没有太多模板的情况下，或者需要向不愿花费时间探究 JavaScript 文件的团队成员公开模板时，建议采用这种方式创建模板。

请注意，还可以在任何元素中(比如在一个 CSS 规则为 `display:none` 的 `div` 元素)创建模板，但这可能会导致意外的错误。

2. 使用 `$.template()` 方法创建模板

此外，还可以在 JavaScript 文件中直接使用 `$.template()` 方法创建模板：



```

$.template('comics', '<div class="comic">
<div class="details"><div class="title">
<h3>${title}</h3></div><div class="year">${year}</div>
<div class="number">${number}</div></div></div>');

```

代码片段 `template-method.txt`

这两个例子都使用了相同的模板格式，变量都封装在 `${}` 块之中。不同之处在于，在使用 `$.template()` 方法时，表示模板的字符串是作为第二个参数传入该方法的。第一个参数是模板的 `id`，它与第一个例子中的 `<script>` 标记中的 `id` 属性类似。

由于 `$.template()` 方法没有清晰地分离行为和结构，因此仅当网站或应用程序中具有多个模板时，或者想利用浏览器缓存模板的定义时，才建议采用 `$.template()` 方法创建模板。之前介绍的第一种方法，是利用特定格式的脚本块来创建模板，这种模板是内联地包含在标记代码之中的，每次当一个页面加载时，都会下载并解析这些脚本块。

将模板放在脚本文件中，可以使模板只下载一次，然后对于后续的页面请求，浏览器可以从缓存中取得模板，这种方式可以节省网络连接和下载的时间。如果模板用于一个单页面应用程序中的个别页面或站点中的一部分时，这可能并不是什么问题。但是，如果有很多应用于大型网站的模板，那么采用缓存方式似乎更胜一筹。

3. 使 `$.template()` 方法具有可维护性

如果需要使用 `$.template()` 方法创建模板，建议采用下面例子中的模式。它使用单个函数来初始化所有应用于站点或应用程序的模板。采用这一模式时，尽管将标记代码保留在

JavaScript 文件中或许并不是最好的解决方案,但它至少使模板保存在单个位置,并且具有更好可维护性。这样,开发人员总是知道模板定义在哪里。下面的代码描述了这一模式。这一基本的模式与第 10 章中讨论的 Garber-Irish 实现类似。在这一模式中,具有一个 `common` 对象用于保存所有页面的通用代码。在 `common` 对象中包含了一个名为 `init` 的方法,还包括了 `init` 方法需要调用以启动应用程序的其他一些方法。在 `init` 方法中,可以通过调用 `myApp.common.templates()` 方法初始化网站的所有模板。



可从
Wrox.com
下载源代码

```
var MyApp = {
  common : {
    init : function(){
      myApp.common.templates();
    },
    templates: function(){
      $.template('comics' ,
        '<div class="comic"></div><div
class="details">
<div class="title"><h3>{title}</h3></div>
<div class="year">{year}</div>
<div class="number">{number}</div></div></div>');
      $.template('author' , '<div class="author">
<div class="name"><h3>{author}</h3></div><div class="bio">
<p>{authorBio}</p></div></div>');
    },
  }
}
```

代码片段 `template-maintenance.txt`

11.1.6 使用 `$.tmpl()` 方法应用模板

在创建好站点的模板之后,就可以开始在页面中使用这些模板。在应用模板时,存在两种截然不同的使用场合。

如果以内联方式创建模板,即将模板放在具有特殊 `type` 的 `<script>` 标记之中。要在页面中应用该模板,只需要将表示该 `<script>` 标记的选择器传递给 `$.tmpl()` 方法即可。下面的代码示例描述了这种应用模板的方法。在下面的例子中,创建了一个 `id` 为 `#comics` 的模板标记。为了在页面中应用该模板,使用 `id` 选择器获取对该模板的一个引用,然后调用 `$.tmpl()` 方法,将 `comics` 数组作为参数传递给该方法。接下来只需要简单地调用 `$.appendTo()` 方法就可以将其插入到文档中。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
```

```

<meta charset="utf-8">
</head>
<body>
<div id="main"> </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script
src="//ajax.aspnetcdn.com/ajax/jquery.templates/beta1/jquery.tmpl.min.js">
</script>
<script>
    $(function(){
        var comics = [{
            imgSrc : "cover1.jpg",
            title : "Captain Templates",
            year : "2010",
            number : "1"
        },
        {
            imgSrc : "cover2.jpg",
            title : "Captain Templates",
            year : "2011",
            number : "2"
        },
        {
            imgSrc : "cover3.jpg",
            title : "Captain Templates",
            year : "2012",
            number : "3"
        }
    ]];
    $("#comics").tmpl(comics).appendTo("#main");
    }
);
</script>
<script id="comics" type="text/x-jquery-tmpl">
    <div class="comic">
    <div class="details">
        <div class="title">
            <h3>{title}</h3>
        </div>
        <div class="year">
            {year}
        </div>
        <div class="number">
            {number}
        </div>
    </div>
    </div>
</script>
</body>
</html>

```

代码片段 inline.html

该页面产生的输出效果如图 11-1 所示。

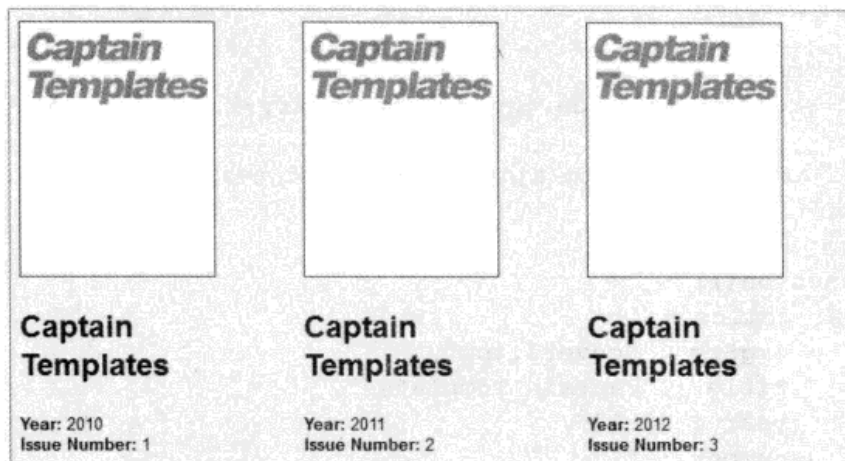


图 11-1

第二种方法是使用编译过的模板。在应用模板时，将\$.template()方法创建的模板名称和一个数据对象作为参数传入 jQuery.tmpl()方法，就可以将模板应用于页面之中，效果与第一种方法完全相同。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<div id="main"> </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script src="
//ajax.aspnetcdn.com/ajax/jquery.templates/beta1/jquery.tmpl.min.js">
</script>
<script>
$(function(){
    var comics = [{
        imgSrc : "cover1.jpg",
        title : "Captain Templates",
        year : "2010",
        number : "1"
    },
    {
        imgSrc : "cover2.jpg",
        title : "Captain Templates",
        year : "2011",
        number : "2"
    },
    {
        imgSrc : "cover3.jpg",
        title : "Captain Templates",

```

```

        year : "2012",
        number : "3"
    }
  ];

  $.template( 'comics' , '<div class="comic">
    <div class="details">
    <div class="title">
    <h3>{title}</h3></div><div class="year">{year}</div>
    <div class="number">{number}</div></div></div>' );
  $.tmpl( "comics",comics ).appendTo( "#main" );
  });
</script>
</body>
</html>

```

代码片段 template-method.txt

11.1.7 在模板中使用远程数据

为了简单起见，本章中的大多数例子都使用了一个在脚本块中定义的 JavaScript 数组，它可以很好地在浏览器中运行示例，不必建立一个 Web 服务器。然而在很多情况下，开发人员需要在模板中使用从 Ajax 请求中获取的动态数据。

在使用 Ajax 数据时，模板的使用方式是相同的。下面是一个使用了 Ajax 请求的简单例子，它说明了如何将 Ajax 数据应用在模板中。

在这个例子，将使用一个名为 data.json 的文件，它的结构如下所示：



```

{
  "comics" : [
    {
      "imgSrc": "cover1.jpg",
      "title": "Captain Templates",
      "year": "2010",
      "number": "1"
    },
    {
      "imgSrc": "cover2.jpg",
      "title": "Captain Templates",
      "year": "2011",
      "number": "2"
    },
    {
      "imgSrc": "cover3.jpg",
      "title": "Captain Templates",
      "year": "2012",
      "number": "3"
    }
  ]
}

```



```

    ]
}

```

代码片段 data.json

在脚本文件中，只需要简单地使用\$.ajax()方法获取数据，然后将数据传递给 success 函数 populate，再在 populate 函数中，使用.tmpl()方法处理 Ajax 请求获取的数据。



可从
Wrox.com
下载源代码

```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<div id="main"> </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script src="
//ajax.aspnetcdn.com/ajax/jquery.templates/betal/jquery.tmpl.min.js">
</script>
<script>
    $(function(){
        var populate = function( data ){
            $( "#comics" ).tmpl(data.comics).appendTo( "#main" );
        }
        $.ajax({
            type : 'get',
            url : 'data.json',
            success : populate,
            dataType : "json"
        });
    });
</script>
<script id="comics" type="text/x-jquery-tmpl">
    <div class="comic">
    <div class="details">
        <div class="title">
            <h3>${title}</h3>
        </div>
        <div class="year">
            <strong>Year:</strong> ${year}
        </div>
        <div class="number">
            <strong>Issue Number:</strong> ${number}
        </div>
    </div>
    </div>
</script>
</body>

```

PDF

</html>

代码片段 ajax.txt

11.1.8 模板标记

在介绍了模板的基础知识之后，下面将进一步介绍 Template 插件的一些高级特性。虽然统一地将数据简单地映射到标记代码中非常有用，但模板的高级特性允许开发人员进行更多的控制。

1. 使用{{if}} {{else}}向模板中添加简单逻辑

{{if}}和{{else}}提供了在模板中根据条件包含内容功能。仅当提供的条件值不为 null 时，它将呈现{{if}}和{{/if}}标记之间的内容。当条件值为 false 时，将呈现{{else}}与{{/if}}标记之间的内容。下面的例子描绘了如何使用{{if}}和{{else}}模板标记，根据条件判断来决定是否包含一个缩略图。如果数据源中没有设置缩略图片，则显示一个默认图片。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<div id="main"> </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script src="
//ajax.aspnetcdn.com/ajax/jquery.templates/beta1/jquery.tmpl.min.js">
</script>
<script>
$(function(){
    var comics = [{
        imgSrc : "cover1.jpg",
        title : "Captain Templates",
        year : "2010",
        number : "1"
    },
    {
        imgSrc : "cover2.jpg",
        title : "Captain Templates",
        year : "2011",
        number : "2"
    },
    {
        title : "Captain Templates",
        year : "2012",
        number : "3"
    }
    ];
```

```

        $( "#comics" ).tmpl( comics ).appendTo( "#main" );
    }
    );
</script>
<script id="comics" type="text/x-jquery-tmpl">
    <div class="comic">
        {{if imgSrc}}
            
        {{else}}
            
        {{/if}}
        <div class="details">
            <div class="title">
                <h3>${title}</h3>
            </div>
            <div class="year">
                ${year}
            </div>
            <div class="number">
                ${number}
            </div>
        </div>
    </div>
</script>
</body>
</html>

```

代码片段 if.txt

将该模板应用于数据源，产生的输出效果如图 11-2 所示。

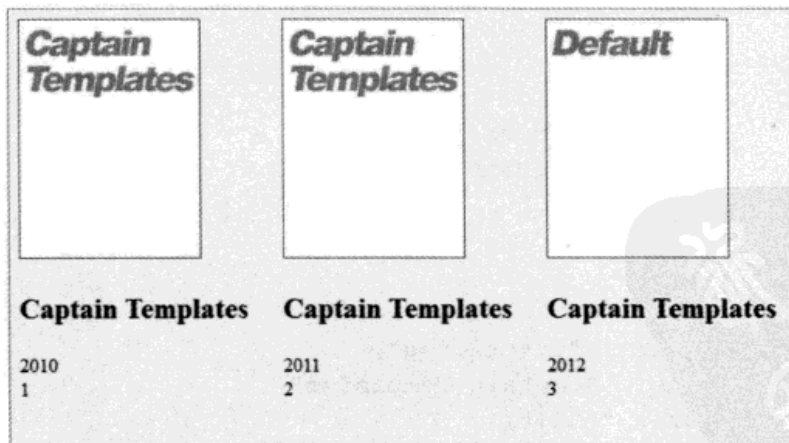


图 11-2

2. 使用{{each}}遍历数据对象

与\$.each()方法类似，{{each}}用于遍历一个数据数组。对于数组中的每一个数据项，它都会把{{each}}与{{/each}}模板标记之间的内容呈现一次。比如在下面的代码示例中，

对于每一个 comic 元素，在它的数据结构中暴露了一个名为 themes 的数组。在相应的内联模板中，对于每一个 comic 模板元素，使用 {{each}} 模板标记列出了它所包含的一个或多个 theme 项的列表。每一个 theme 包装在一个 元素中。



```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<div id="main"> </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script src="
//ajax.aspnetcdn.com/ajax/jquery.templates/betal/jquery.tmpl.min.js">
</script>
<script>
$(function(){
    var comics = [{
        imgSrc : "cover1.jpg",
        title : "Captain Templates",
        themes : [
            "code reuse" ,
            "separation of content and behavior" ,
            "template tags"
        ],
        year : "2010",
        number : "1"
    },
    {
        imgSrc : "cover2.jpg",
        title : "Captain Templates",
        themes : [
            "code reuse" ,
            "moustaches" ,
            "templating for fun and profit"
        ],
        year : "2011",
        number : "2"
    },
    {
        imgSrc : "cover3.jpg",
        title : "Captain Templates",
        themes : [ "threes" ],
        year : "2012",
        number : "3"
    }
    ];
    $( "#comics" ).tmpl( comics ).appendTo( "#main" );
}
```



```

);
</script>
<script id="comics" type="text/x-jquery-templ">
  <div class="comic">
    <div class="details">
      <div class="title">
        <h3>{title}</h3>
      </div>
      <div class="year">
        <strong>Year:</strong> {year}
      </div>
      <div class="number">
        <strong>Issue Number:</strong> {number}
      </div>
      <div class="themes">
        <strong>Themes</strong>
        <ul>
          {{each themes}}
            <li>{value}</li>
          {{/each}}
        </ul>
      </div>
    </div>
  </div>
</script>
</body>
</html>

```

代码片段 each.txt

在\$.tmpl()方法中，将模板应用于数据，产生的输出效果如图 11-3 所示。

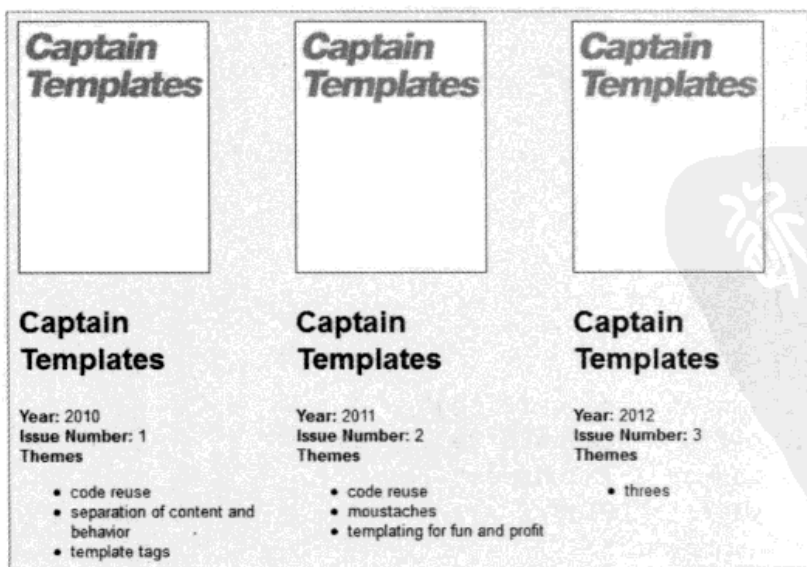


图 11-3

此外, `{{each}}` 还暴露了一个 `index` 属性, 它允许在模板自身中集成一个计数器。下面的代码修改了前面的模板, 演示了如何使用 `index`:



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<div id="main"> </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script src="
//ajax.aspnetcdn.com/ajax/jquery.templates/beta1/jquery.tmpl.min.js">
</script>
<script>
$(function(){
    var comics = [{
        imgSrc : "cover1.jpg",
        title : "Captain Templates",
        themes : [
            "code reuse" ,
            "separation of content and behavior" ,
            "template tags"
        ],
        year : "2010",
        number : "1"
    },
    {
        imgSrc : "cover2.jpg",
        title : "Captain Templates",
        themes : [
            "code reuse" ,
            "moustaches" ,
            "templating for fun and profit"
        ],
        year : "2011",
        number : "2"
    },
    {
        imgSrc : "cover3.jpg",
        title : "Captain Templates",
        themes : [ "threes" ],
        year : "2012",
        number : "3"
    }
    ];
    $( "#comicsIndex" ).tmpl( comics ).appendTo( "#main" );
    }
);
</script>
```



```

<script id="comicsIndex" type="text/x-jquery-templ">
  <div class="comic">
    <div class="details">
      <div class="title">
        <h3>{title}</h3>
      </div>
      <div class="year">
        <strong>Year:</strong> {year}
      </div>
      <div class="number">
        <strong>Issue Number:</strong> {number}
      </div>
      <div class="themes">
        <strong>Themes</strong>
        {{each themes}}
          <strong>${$index + 1}</strong>. ${$value}
        {{/each}}
      </div>
    </div>
  </div>
</script>
</body>
</html>

```

代码片段 `each-index.txt`

在\$.tmpl()方法中，将模板应用于数据，产生的输出效果如图 11-4 所示。

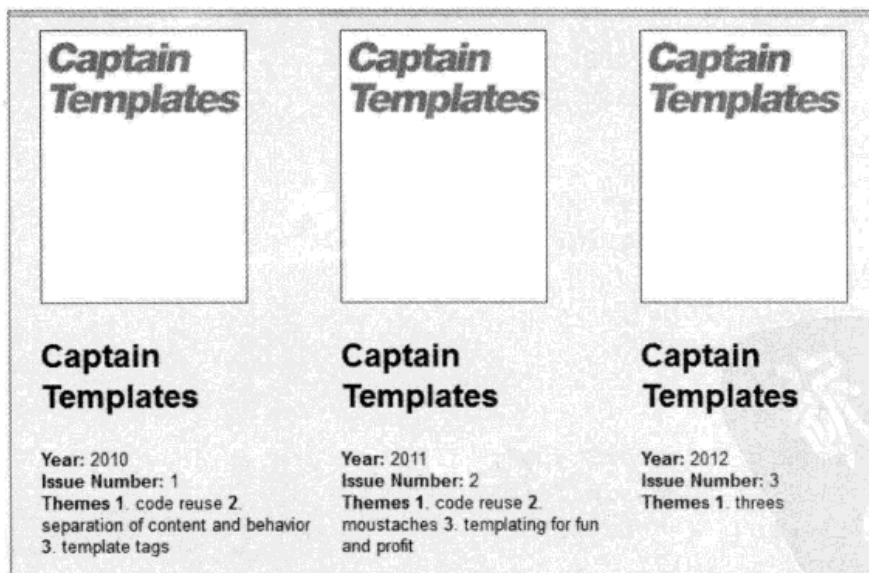


图 11-4

3. 使用{{tmpl}}嵌套模板

{{tmpl}}模板标记支持模板的嵌套。它允许在更深的程度上实现代码重用。例如，如

果具有一个通用的结构，用于在多个独立的模板中显示缩略图和标题，那么可以将缩略图和标题定义为一个模板，并在任何需要的地方引用该模板即可。

下面三个模板都是内联模板。第一个模板 `headerTemplate` 定义了一个通用的头部，可以用于其他两个模板——即 `comicsConcise` 模板和 `comicsExtended` 模板。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<div id="main"> </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script src="
//ajax.aspnetcdn.com/ajax/jquery.templates/betal/jquery.tmpl.min.js">
</script>
<script>
$(function(){
    var comics = [{
        imgSrc : "cover1.jpg",
        title : "Captain Templates",
        themes : [
            "code reuse" ,
            "separation of content and behavior" ,
            "template tags"
        ],
        year : "2010",
        number : "1"
    },

    {
        imgSrc : "cover2.jpg",
        title : "Captain Templates",
        themes : [
            "code reuse" ,
            "moustaches" ,
            "templating for fun and profit"
        ],
        year : "2011",
        number : "2"
    },

    {
        imgSrc : "cover3.jpg",
        title : "Captain Templates",
        themes : [ "threes" ],
        year : "2012",
        number : "3"
    }
    ]
});
```



```

    });
    $( "#comicsConcise" ).tmpl( comics ).appendTo( "#main" );
    $( "#comicsExtended" ).tmpl( comics ).appendTo( "#main" );
  }
);
</script>
<script id="headerTemplate" type="text/x-jquery-tmpl">
  <header>
    <h1>${title}</h1>
    
  </header>
</script>
<script id="comicsConcise" type="text/x-jquery-tmpl">
  <section class="comic">
    {{tmpl "#headerTemplate"}}
    <div class="details">
      <div class="year">
        <strong>Year:</strong> ${year}
      </div>
      <div class="number">
        <strong>Issue Number:</strong> ${number}
      </div>
    </div>
  </section>
</script>
<script id="comicsExtended" type="text/x-jquery-tmpl">
  <section class="comic extended">
    {{tmpl "#headerTemplate"}}
    <div class="details">
      <div class="year">
        <strong>Year:</strong> ${year}
      </div>
      <div class="number">
        <strong>Issue Number:</strong> ${number}
      </div>
      <div class="themes">
        <strong>Themes</strong>
        <ul>
          {{each themes}}
            <li>${$value}</li>
          {{/each}}
        </ul>
      </div>
    </div>
  </section>
</script>
</body>
</html>

```

代码片段 tmpl.txt

充分利用嵌套模板的功能，可以将模板分解为更小的代码块，这样可以使模板代码具有更好的可维护性。找出通用的元素，将它们分离到一个独立的模板中，这样，当需要修改通用元素时，只需要修改单个模板即可，不必在多个模板中寻找需要修改的代码。

4. 使用{{html}}在模板中嵌入 HTML

有时，在从 Web 服务器返回的数据中，包含了整段的 HTML。使用{{html}}模板标记，可以将这些预生成的 HTML 直接插入到模板中。如果不使用{{html}}，诸如jquery.com这样的文本将按照 HTML 字面值来呈现，比如>a href=...，因此无法获取预期的效果。

在下面的代码中，数据源中包含了一个 description 字段，该字段中包含了 HTML 文本。在直接输出 HTML 片段的区域，使用{{html}}模板标记进行了封装。



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<div id="main"> </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script src="
//ajax.aspnetcdn.com/ajax/jquery.templates/beta1/jquery.tmpl.min.js">
</script>
<script>
$(function(){
    var comics = [{
        imgSrc : "cover1.jpg",
        title : "Captain Templates",
        themes : [
            "code reuse" ,
            "separation of content and behavior" ,
            "template tags"
        ],
        year : "2010",
        number : "1",
        description : "<p>In this thrilling origin issue Captain
Templates saves the day by using templates instead of a bunch
of <strong>awkward</strong> string concatenation</p>"
    },
    {
        imgSrc : "cover2.jpg",
        title : "Captain Templates",
        themes : [
            "code reuse" ,
            "moustaches" ,
            "templating for fun and profit"
        ]
    }
    ]
});
```



```

    ],
    year : "2011",
    number : "2",
    description : "<p>In battling his <em>arch nemesis</em>
<strong>Doctor Plus Sign</strong> Captain Templates falls
Into a coma. </p><p>A thrilling issue with a cliffhanger
ending</p>"
  },
  {
    imgSrc : "cover3.jpg",
    title : "Captain Templates",
    themes : [ "threes" ],
    year : "2012",
    number : "3",
    description : "<p>Captain Templates awakens from his coma
and defeats the evil <strong>Doctor Plus Sign</strong></p>"
  }
];
$( "#comics" ).tmpl( comics ).appendTo( "#main" );
}
);
</script>
<script id="comics" type="text/x-jquery-tmpl">
  <section class="comic">
    <header>
      
      <h3>{title}</h3>
    </header>
    <div class="details">
      <div class="year">
        <strong>Year:</strong> {year}
      </div>
      <div class="number">
        <strong>Issue Number:</strong> {number}
      </div>
      <div class="description">
        {{html description}}
      </div>
    </div>
  </section>
</script>
</body>
</html>

```

代码片段 html.txt

该代码产生的输出效果如图 11-5 所示。

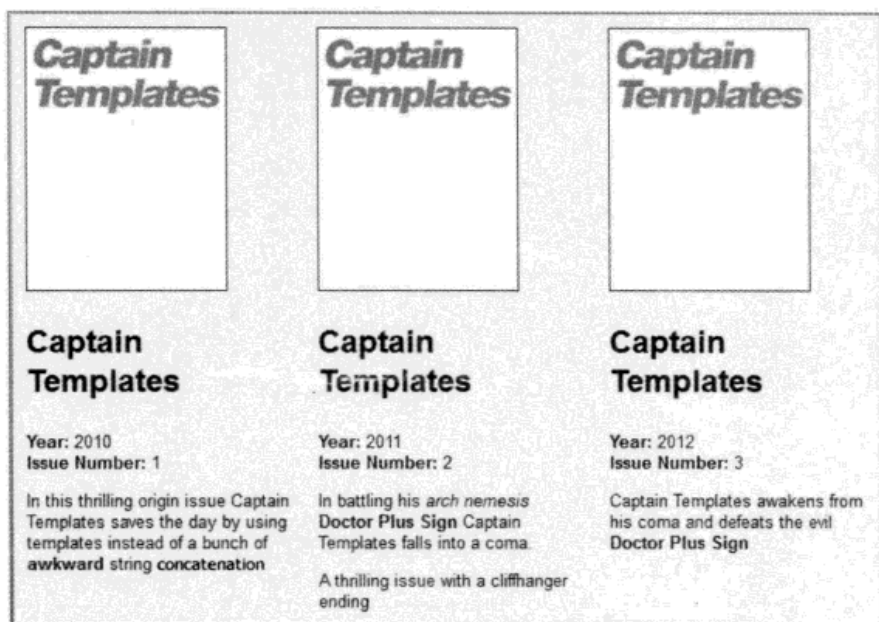


图 11-5

11.2 小结

经过本章的学习，我们已经理解了使用模板系统分离内容和行为的优势。本章详细介绍了在 jQuery 中，如何使用已有的 jQuery Template 插件来实现模板，以及如何充分利用模板标记以便更精确地控制数据的呈现。

本章还介绍了 jQuery Template 插件的现状，并介绍了模板技术未来的发展方向。无论何时，当新版本的插件可用时，你已经为此做好了准备。



第 12 章

编写 jQuery 插件

本章内容

- 插件基础
- 插件的最佳模式和实践
- jQuery UI Widget Factory

本章将介绍如何编写 jQuery 插件。一个顶级 jQuery 开发人员必须具备这样的基本技术：能够使用用户自定义方法扩展 jQuery。

jQuery 之所以盛行，很大程度上来源于开发社区驱动的插件生态系统，以及可以对 jQuery 核心功能进行广泛的扩展。

从 `hoverIntent` 插件对用户界面提供的细致增强，到 jQuery UI 项目中那些功能完备的组件，jQuery 中的插件是一种生机勃勃的重要资源。在 jQuery 中参与编写插件或创建自己的插件都很简单。本章将介绍如何编写 jQuery 插件。

本章将介绍扩展 jQuery 的多种不同方法、通用的插件规范，以及可以进一步扩展插件的高级选项和模式。

在学习完本章之后，你将具备自行扩展和增强 jQuery 的能力，几乎在编写任何插件时，都可以找到丰富的资源并从中获得帮助，也许你应该亲自体验一下插件的开发过程。

12.1 插件基础

在最基本的层面上，创建一个 jQuery 插件非常简单！在学习完本章的前几页内容之后，你就具备足够的知识，可以创建一个简单的 jQuery 插件了。

但是，请不要被最低层面上创建 jQuery 插件的简单性所迷惑。可以很“简单”地创建一个基础的 jQuery 插件，并不意味着正确地创建 jQuery 插件是件容易的事情。在创建 jQuery

插件时，遵循一些最佳实践可以确保你的 jQuery 插件具有优化的性能、更少的 bug，并为 jQuery 插件与代码运行的 jQuery 环境之间提供高度的互操作性。本章第一部分将介绍 jQuery 插件的一些重要的基础知识。

12.1.1 遵循 jQuery 插件的命名规范

假如在学习完本章之后，你已经完全具备了 jQuery 插件开发的相关知识，并愉快地准备编写自己的第一个 jQuery 插件。打开你最喜欢的编辑器或 IDE，创建一个新的脚本文件。根据你所使用的编辑器，可能已经具有一个创建文件名的选项，或者直到你保存文件时才会出现这样的选项。无论使用哪一种编辑器，我们事先都需要知道如何命名 jQuery 插件的脚本文件。答案非常简单，但事先应该遵循 jQuery 插件的命名规范。

通常情况下，jQuery 插件采用下面的模式进行命名：

```
jquery.pluginName.js
```

min 版也采用与之类似的命名规范，并添加一个 min 标记：

```
jquery.pluginName.min.js
```

在命名 jQuery 插件时包含 jquery 这一名称，原因很简单——这是一个 jQuery 插件，因此在文件名中指出它的兼容性很方便。某些插件支持多种 JavaScript 库，因此这是一个重要的说明。

在 min 版本的文件名中添加“.min.”，可以使两个版本同时存在，这一命名方式非常方便，因为在调试代码时，它允许开发人员在 full 版和 min 版之间进行切换。

12.1.2 如何扩展 jQuery

在介绍了如何命名 jQuery 插件的脚本文件之后，接下来将学习如何实际地扩展 jQuery。

jQuery 提供了两种独立而简单的模式和一个稍微复杂但非常强大的选项，用于扩展 jQuery。

第一种扩展 jQuery 的方法，最常见的是通过一个简单的、jQuery 函数 prototype 属性的别名(jQuery.fn)进行扩展。第二种扩展 jQuery 的方式是采用 jQuery.extend()方法。第三种更加复杂的办法，是使用强大的 jQuery UI Widget Factory 进行扩展。

1. 使用 jQuery.fn

如果你曾经查看过 jQuery 的源代码，那么早就在源代码中看到过下面这样的代码。在 jQuery 1.7.1 版本的源代码中，它位于第 97 行：



可从
Wrox.com
下载源代码

```
jQuery.fn = jQuery.prototype = { //jquery goes here //}
```

代码片段 jquery.fn.txt

也许你对原型继承(prototypal inheritance)的概念还不熟悉,它是 JavaScript 继承的关键元素。

简而言之,JavaScript 对象的原型是另外一个对象,它包含了该类型对象原生的属性和方法。核心 JavaScript 原生对象都具有自己的 prototype 属性。下面的例子说明了原生 prototype 属性的常见用法。在这个例子中,对 Array.prototype 进行了增强,弥补了在非兼容浏览器之中遗漏的 EcmaScript 功能。运行下面的代码,将为包含该代码的页面中创建的所有数组添加一个 [].every() 方法,包括过去已经创建过的数组和将来创建的数组。



```
//from
//https://developer.mozilla.org/en/JavaScript/Reference/Global_Ob
//jects/Array/every

if (!Array.prototype.every){
  Array.prototype.every = function(fun /*, thisp */){
    "use strict";
    if (this === void 0 || this === null) {
      throw new TypeError();
    }
    var t = Object(this),
        len = t.length >>> 0;
    if (typeof fun !== "function"){
      throw new TypeError();
    }
    var thisp = arguments[1];
    for (var i = 0; i < len; i++){
      if (i in t && !fun.call(thisp, t[i], i, t)){
        return false;
      }
    }
    return true;
  };
}
```

代码片段 every.txt

值得注意的是,fn/prototype 是“实时”的。无论何时,只要在某个类型的原型中增加了属性或方法,任何从该类型创建的对象就可以立即访问这些新增的属性或方法。JavaScript 的点号操作符(.)具有特殊的功能,它将从点号左侧的表达式中搜索点号右侧的关键字,遍历“原型链”直到找到一个匹配项。对象实例都具有各自的内部“类”作为它们的原型,因此对于原型的改变将“实时”地反映到对象实例上。正是出于这样的原因,在页面的生命周期中,开发人员可以在某处创建一个 jQuery 的实例,然后再通过随后才加载的脚本为 jQuery 添加特性和功能——甚至在几秒钟或几分钟之后。

jQuery.fn 最基本的使用方法如下所示:



可从
Wrox.com
下载源代码

```
jQuery.fn.newStuff = function() {
    console.log("It's full of (javascript) stars");
};
```

代码片段 *basic-fin-plugin.txt*

虽然这个初级的例子并不完美，但它是将新功能添加到 jQuery 所需的最少工作。在后面的小节中，将增强并改进这种编写 jQuery 插件的模式。

2. 使用 \$.extend()

除了使用 JavaScript 的原型继承模式之外，jQuery 还提供了一个名为 \$.extend 的工具方法，可以使用 \$.extend 方法来增强 jQuery。在第 3 章中曾经介绍过，\$.extend 方法最基本的功能就是合并两个对象。当调用 \$.extend 方法时，如果只传入一个参数，那么它将把该参数对象合并到 jQuery 中。

下面的代码为 jQuery 扩展了一个名为 newStuff 的方法：



可从
Wrox.com
下载源代码

```
jQuery.extend({
    newStuff: function() {
        console.log("It's full of (javascript) stars");
    }
});
```

代码片段 *plugin-extend.txt*

3. jQuery UI Widget Factory

Widget Factory 是一个 jQuery 方法，它接收两个或者三个参数：第一个参数是一个名称空间；第二个参数是一个已有的 widget 原型，它将从该原型进行继承；第三个参数是一个可选的对象字面量，它作为新的 widget 原型。

在 jQuery UI 项目的幕后，正是使用了 Widget Factory，它被设计用于通过统一的 API 来创建复杂的、具有状态的插件。

Widget Factory 最简单的实现只需要一行代码，比如：



可从
Wrox.com
下载源代码

```
$.widget('namespace.newStuff', {});
```

代码片段 *simple-plugin-factory.txt*

在上面这行代码中，调用了 \$.widget 方法，传入了一个名称空间(namespace.newStuff) 和一个作为原型的空对象。当然这样的代码没有什么实际的功能，但这是一个开始。

在本章后面的“深入 Widget Factory”小节中将更详细地介绍 Widget Factory 的特性。

4. 选用哪一种模式创建 jQuery 插件?

以上三种方法都可以创建 jQuery 插件,你可以根据所编写 jQuery 插件的需要,选择你最喜欢的方式来创建 jQuery 插件。本章后面还将更深入地探讨 jQuery 插件开发模式的选择问题。对于刚开始涉足 jQuery 插件开发工作的程序员,前两种方式更适合作为起点。

为了简单起见,本章后面的绝大多数例子都将使用以 JavaScript 为中心的 jQuery.fn 方法来编写 jQuery 插件,这种方式最为直观。

12.1.3 jQuery 插件通用指南

上面的小节已经介绍了扩展 jQuery 的基础知识,接下来将介绍一些编写 jQuery 插件的通用指南,遵循这些指南将有助于编写出正确的 jQuery 插件。jQuery 插件开发的基本原则,绝大部分与开发 jQuery 应用程序、网站或组件的原则类似。但在编写 jQuery 插件时,还有一些应该注意的关键区别。这一小节将简要地介绍一些编写 jQuery 插件的通用指南。

1. 记住 this 关键字

在从基础 jQuery 应用进入到高级 jQuery 开发时,开发人员常常犯的一个错误就是:忘记了在 jQuery 插件的上下文中, this 关键字所指向的对象有所不同。在 jQuery 开发中, this 关键字通常引用的是当前正在操作的 DOM 元素。但在当前的 jQuery 插件上下文中, this 关键字引用的是当前 jQuery 实例自身。因此在 jQuery 插件体内,类似于下面这样的代码:



可从
Wrox.com
下载源代码

```
$(this).toggle()
```

代码片段 *extra-this.txt*

实际上可以直接写为:



可从
Wrox.com
下载源代码

```
this.toggle()
```

代码片段 *simplified-this.txt*

在 jQuery 插件体内。将 this 关键字包装在 \$() 之中,实际上等效于执行 `$("#element")` 语句。

唯一的例外就是当在当前 jQuery 集合中遍历所有元素时。在 \$.each 循环的循环体内, this 关键字引用的是当前这一轮遍历时所暴露的 DOM 元素。下面的代码示例说明了这一区别:



可从
Wrox.com
下载源代码

```
(function($){  
    //插件定义
```

```
$.fn.pinkify = function() {
    //this 引用的是 jQuery 本身，紧接着调用了 each() 方法
    return this.each(function() {
        //在循环体内，this 关键字引用的是 DOM 元素
        $( this ).css({
            "background" : "#fe57a1",
            "color" : "#fff",
            "text-shadow" : "none"
        })
    });
};
})( jQuery );
```

代码片段 remember-this.txt

2. 总是确保\$指向 jQuery

在编写 jQuery 插件时，常常用到一个最简单的代码片段，这个代码片段也是编写 jQuery 插件时最重要的代码片段之一。开发人员必定使用到一个立即执行的函数表达式，并将 \$ 绑定到 jQuery。总是存在这样的可能，即某些其他 JavaScript 库(比如常见的 PrototypeJS)或者其他的 JavaScript 代码已经重新编写了 \$，将其指向其他的值，因此在编写 jQuery 插件时确保 \$ 总是指向 jQuery 是至关重要的。

下面的代码说明了如何确保 \$ 总是指向 jQuery。它将插件的定义包装在一个函数表达式中。该函数接收一个参数 \$，由于该函数是立即执行的，因此将 jQuery 作为参数传入该函数，就可以确保 \$ 总是绑定于 jQuery。



可从
Wrox.com
下载源代码

```
(function($) {
    $.fn.newStuff = function() {
        /*无论在全局名称空间中$包含了什么样的值，
        *在该函数内$总是表示 jQuery
        */
    };
})( jQuery );
```

代码片段 jquery-is-jquery.txt

另外，该模式还有一个常见的增强，值得开发人员学习。它为 window 和 document 对象创建了一个局部引用，并确保 undefined 保持 undefined 的值。增强模式具有更多的优点。

为 window 和 document 对象创建了一个局部引用，可以使这两个对象在函数体内最精简(minified，以字节计算)，还可以缩短要查询的表，从而在函数体内加快对这些对象的访问速度。

另外，由于在低版本的 ECMAScript 标准中可以对 undefined 重新赋值，如果有人无意

或者恶意地重新定义 `undefined` 的值，这就会成为潜在 bug 的来源。将 `undefined` 定义为一个参数，然后在传入参数时不传入这个参数，这是一种聪明的办法，可以确保 `undefined` 正是我们所期望的 `undefined` 值。



可从
Wrox.com
下载源代码

```
(function( $, window, document, undefined ) {
$.fn.newStuff = function() {
    /* 无论在全局名称空间中$为何值,
    * 在这里$都表示 jQuery
    */
};
})( jQuery, window, document );
```

代码片段 `window-document.-undefined.txt`

3. 正确地使用分号

如果有人使用了你编写的 jQuery 插件，作为开发人员应该感到高兴。但请注意一个重要问题，即使用者最后可能会将你的脚本与其他代码合并在一起并最小化。因此，在开发 jQuery 插件时必须注意正确地使用分号。

如果遵循良好的编码实践进行开发，那么你可能已经正确地使用了分号。但当你的代码将在未知的环境中运行时，这一问题尤为重要。

可以参考 Google JavaScript Style Guide 中关于分号的说明(<http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml#Semicolons>): “依赖于隐式插入的分号可能会导致狡诈的、难以调试的问题。完全没有必要这样，请明确地使用分号。”

简而言之——分号是你的好朋友！

另外一个有关分号的常用实践，就是在定义插件的代码之前添加一个前导分号，这一技巧可以确保在最小化和合并脚本的情况下，插件的代码可以正常运行。在最小化脚本代码时，由于其他代码未能正确使用分号或者存在问题的语句，都可能会导致其他代码的中断。在定义插件的代码之前插入一个前导分号，可以确保任何之前的语句都在遇到这个分号时显式地结束。



可从
Wrox.com
下载源代码

```
; (function( $, window, document, undefined ) {
$.fn.newStuff = function() {
};
})( jQuery, window, document );
```

代码片段 `leading-semicolon.txt`

4. 尽可能地返回 jQuery 对象

jQuery 就是 Fluent Interface 的一个例子，Fluent Interface 是由 Martin Fowler 和 Eric Evans 提出的一种软件设计模式，Fowler 在 2005 发表的一篇文章中提出了这一模式 (<http://www.martinfowler.com/bliki/FluentInterface.html>)。它的基本概念包括将方法与读起来更像自然语言的 API 链接在一起，以创建更加自然、更容易访问的 API。在描述它的一个实现时，Fowler 说“API 首先被设计为更容易阅读、更流畅”。jQuery 完全符合这一描述。

请记住，jQuery 的关键特性之一就是支持链式方法调用。链式方法调用为 jQuery API 提供了简洁、流畅的编程体验，它正是吸引众多开发人员投入 jQuery 库怀抱的特性之一。对于 jQuery 库和其他或大或小的 JavaScript 库而言，链式方法调用都是一个流行的特性。

在 jQuery 插件的编程实践中，要支持链式方法调用，就是要尽可能地返回一个指向 jQuery 对象的 this 对象。由于插件运行在与页面脚本代码相同的环境之中，因此在插件中尽可能地返回 jQuery 对象是至关重要的。在绝大多数情况下，你编写的 jQuery 插件将遵循下面代码示例中描述的模式。在插件直接的作用域中，它将返回 this 对象并维护该对象完整的集合，以便可以将该对象继续传递给其他 jQuery 方法。



可从
Wrox.com
下载源代码

```
(function( $ ) {
    $.fn.pinkify = function() {
        return this.css({
            "background" : "#fe57a1",
            "color" : "#fff",
            "text-shadow" : "none"
        });
    };
})( jQuery );
```

代码片段 chaining.txt

这一编写插件的模式是最佳实践，也是首选的方法。然而有时并不总是能返回 jQuery 对象。一些例外的情况值得注意。虽然我们总是希望能够链式调用方法，但某些情况下会中断调用链。

某些类型的方法会中断链式调用，比如那些不得不返回一个特定值的方法。jQuery 自身也包含了一些这样的例子。这些方法的返回值是一个度量值，比如 \$.height(至少在作为一个 getter 调用时)、一个布尔值，比如 \$.isArray，或者其他类似于 \$.type(它返回一个字符串，指出一个 JavaScript 对象内部的 [[Class]])的方法。下面的代码描述了另外一种例外情况。该插件返回一个 Boolean 值，指出一个元素的尺寸是否大于一组指定的维度值。



可从
Wrox.com
下载源代码

```
(function($) {
    $.fn.isWider = function( width ) {
        return this.width() > width ;
    };
});
```

```
})( jQuery );
```

代码片段 iswider.txt

5. 遍历对象

在jQuery插件开发中,常常需要遍历一个集合中的所有对象,并对每一个成员执行相应的操作。通常使用\$.each 来实现对集合的遍历。下面的代码描述了这一常见模式,它返回 this.each 的结果:



可从
Wrox.com
下载源代码

```
function( $ ) {
    $.fn.randomText = function() {
        return this.each(function() {
            $( this ).text( Math.random * 1000 ).show()
        })
    };
})( jQuery );
```

代码片段 randomTxt.txt

尽管类似的代码很常见,但重要的是请注意在某些情况下,不必使用\$.each。比如下面的例子,使用\$.each 来遍历集合就是多余的,因为\$.css 本身可以对集合中的每一个元素项执行操作。



可从
Wrox.com
下载源代码

```
function( $ ) {
    $.fn.pinkify = function() {
        return this.each(function() {
            $( this ).css({
                "background" : "#fe57a1",
                "color" : "#fff",
                "text-shadow" : "none"
            });
        });
    };
})( jQuery );
```

代码片段 extra-each.txt

可以将上面的代码重写为下面的形式:



可从
Wrox.com
下载源代码

```
function( $ ) {
    $.fn.pinkify = function() {
        return this.css({
            "background" : "#fe57a1",
```



```

        "color" : "#fff",
        "text-shadow" : "none"
    });
};
})( jQuery );

```

代码片段 *return-css.txt*

很多 jQuery 方法都返回一个完整的集合，因此当创建 jQuery 插件时请充分利用这一特性。特别是在想把代码提交给开发社区的情况下，更应该充分利用 jQuery 提供的特性。不积跬步，无以至千里。

12.1.4 jQuery 插件最佳实践

在下面的小节中，将介绍一些 jQuery 插件开发的最佳实践，这些最佳实践是 jQuery 社区在过去几年归纳总结的。像任何其他建议一样，这些最佳实践并非适用于你所编写的每一行代码，但是理解这些最佳实践的原则，既有利于你自己的开发工作，也有助于更好地理解其他插件开发者的工作。

1. 使用 jQuery 插件模式

在 Mike Alsup 发表的文章 *A Plugin Development Pattern* 一文中(<http://www.learning-jquery.com/2007/10/aplugin-development-pattern>)，系统地提出了开发 jQuery 插件时应遵循的一些最佳实践。尽管这篇文章发表于几年之前，但它依然具有重要价值，也是众多 jQuery 插件开发者常常引用的重要资源。下面将详细地介绍这些最佳实践的模式。

2. 在 jQuery 名称空间中只为插件声明单个名称

假如有一些可用的插件，并且这些插件都聚集了一些通用的功能，那么插件的名称很可能会出现相互冲突的问题。为了尽可能地避免名称冲突，只为你的插件声明单个名称空间是非常有效的。例如，请不要像下面的代码那样定义多个与插件有关的方法：



可从
Wrox.com
下载源代码

```

jQuery.fn.bestPluginEverInit = function() {
//init
};
jQuery.fn.bestPluginEverFlip = function() { {
//flip
};
jQuery.fn.bestPluginEverFlop = function() { {
//flop
};
jQuery.fn.bestPluginEverFly = function() { {
//fly
};
};

```

代码片段 *toomanynames.txt*

实际上, 应该为插件创建单个入口点, 并以更加优雅的方式来暴露其他功能。



可从
Wrox.com
下载源代码

```
jQuery.fn.bestPluginEver = function() {
    // 单个名称空间
}
```

代码片段 *single-namespace.txt*

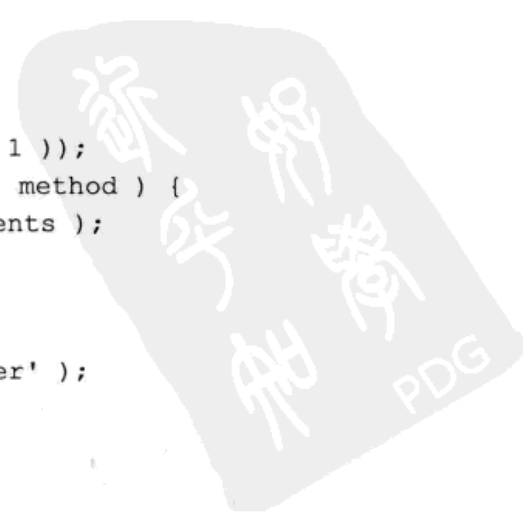
这一最佳实践的出发点, 就是使用一个方法查询表向公开的方法调用点(public endpoint)暴露内部的方法。通过将所有方法封装在插件的闭包(closure)之中, 这一模式保持了一个清晰的名称空间。要调用内部的方法, 只需要将方法名称字符串作为第一个参数传入即可, 该方法所需的其他任何参数, 都可以作为第二个参数传入。在 jQuery 插件的开发社区中, 这种类型的方法封装和架构是一种标准, 无数插件都使用了这种模式, 包括 jQuery UI 项目中的插件和 widgets。下面的代码描述了这一基本模式:



可从
Wrox.com
下载源代码

```
(function( $ ){
    var methods = {
        init : function( options ) {
            //init
        },
        flip : function( howMany ) {
            //flip
        },
        flop: function( ) {
            //flop
        },
        fly : function( ) {
            //fly
        }
    };

    $.fn.bestPluginEver = function( method ) {
        if ( methods[method] ) {
            return methods[ method ].apply( this,
                Array.prototype.slice.call( arguments, 1 ));
        } else if ( typeof method === 'object' || ! method ) {
            return methods.init.apply( this, arguments );
        } else {
            $.error( 'Method'
                + method
                + ' does not exist in the bestPluginEver' );
        }
    };
})( jQuery );
```



```
//调用 init 方法
$( 'div' ).tooltip();

//调用 fly 方法
$( 'div' ).tooltip('fly');

// 调用 flip 方法并传入一个参数
$( 'div' ).tooltip('flip' , 'twice');
```

代码片段 *single-namespace-pattern.txt*

3. 为插件中的事件和数据定义名称空间

除了在 jQuery 名称空间中为插件声明单个名称空间之外, 为插件的事件和数据声明名称空间也同样重要。对于事件和数据, 问题是内在的因此不是那么明显。但这绝不意味着这一问题并不重要。

为此, 当建立事件时, 请充分利用添加事件名称空间的功能来绑定事件。这一简单的技术允许开发人员安全地解除事件绑定, 不会错误地解除插件之外的代码在同一元素上绑定的事件。

要为事件定义名称空间, 只需要在要绑定的事件名称之后追加一个.namespace 即可。下面的代码描述了这一模式, 它在 window 对象的 resize 事件上设置了一个名称空间:



```
(function( $ ){
    var methods = {
        init : function( options ) {
            $( window ).bind( 'resize.bestFunctionEver', methods.flop );
        },
        destroy : function() {
            $( window ).unbind( 'resize.bestFunctionEver' );
        },
        flip : function( howMany ) {
            //flip
        },
        flop: function() {
            //flop
        },
        fly : function() {
            //fly
        }
    };

    $.fn.bestPluginEver = function( method ) {
        //调用方法的代码
    }
});
```



```
};
})( jQuery );
```

代码片段 `event-namespace.txt`

与之类似，如果插件需要存储数据，那么最好将数据存储在单个 `$.data` 项之中。如果需要存储多项数据，应该采用一个对象字面量来包含所有必需的数据，并将其存储在单个 `$.data` 项中。

下面的代码示例说明了如何用 `$.data` 来存储一个对象字面量：



可从
Wrox.com
下载源代码

```
var data = this.data('bestPluginEver');
if (!data) {
    this.data('bestPluginEver', {
        "color" : "blue",
        "title" : "my dashboard",
        "width" : "960px"
    });
}
```

代码片段 `namespaced-data.txt`

如果开发人员遵循这些简单的指南，就可以显著减少名称空间的冲突。这将大大减少开发人员在编写插件时可能遇到的各种麻烦。

4. 接收一个 Options 参数以控制插件的行为

对于一个开发人员，没有什么比在调用一个函数或方法时，要求记住各个参数的顺序更令人沮丧的事情了——“这些代码将在 `setTimeout` 方法中指定的毫秒数之前执行，还是之后执行呢？”如果只有一两个参数可能没什么问题，但如果参数较多，就必须编写说明文档指出哪一个参数该放在什么位置。

另外，在具有多个可选参数的情况下，调用包含多个空参数的函数，会让人感到混淆不清，比如下面的代码：



可从
Wrox.com
下载源代码

```
soManyOptions("#myDiv", , , 14 , "#ff0000" );
```

代码片段 `empty-arguments.txt`

没人愿意处理这样的代码。它不但容易让人感到混淆(这是第 5 个参数，还是第 4 个数？)，而且说实话，这样的代码很丑陋。

为了缓解这种情况，建议开发人员使用一个对象作为可选的参数，当调用插件时，可以使用该可选参数来扩展默认设置。下面的代码说明了如何接收一个可选参数，并使用

`$.extend` 将可选参数与插件的默认设置进行合并。

这是一种很好的方法，可以为插件提供复杂的配置选项。



```
(function( $ ){
    $.fn.bestPluginEver = function( options ) {
        var settings = $.extend( {
            "color" : "blue",
            "title" : "my dashboard",
            "width" : "960px"
        }, options);
        return this.each(function() {
            //best.plugin.ever
        });
    };
})( jQuery );

$( 'div' ).bestPluginEver({
    'color' : 'AliceBlue'
});
```

代码片段 `configuration-object.txt`

5. 为插件的默认设置提供公开的访问

根据前面的例子，直接暴露默认的插件设置也是有益的。这样，不必调用插件就可以直接设置插件的默认选项。对于插件的使用者而言，这一特性可以让他们以最少的代码定制插件的实现。下面的代码示例描述了这一情况，它使用 `$.extend` 以类似的办法扩展了插件的默认选项，但也同时直接暴露了 `$.fn.bestPluginEver.defaults` 对象。在前面的章节中已经介绍过，`$.fn` 是 `jQuery.prototype` 的一个别名，它是一个“实时(live)”的链接。任何对 `$.fn.bestPluginEver.defaults` 的修改，都会作用于在对默认对象进行修改之前或之后创建的任何插件的实例。这与 `options` 参数的作用域不同，`options` 参数仅仅影响到具体的实例。

值得注意的是，在该例中使用了一个空对象作为传入 `$.extend` 方法的第一个参数。这可以确保当 `options` 参数被调用时，原型化的默认对象依然完好无缺。这一模式既支持扩展以更新单个实例，也保留了通过重写默认原型以设置插件的全局默认值的能力。



```
(function( $ ){
    $.fn.bestPluginEver = function( options ) {
        var settings = $.extend( {}, $.fn.bestPluginEver.defaults,
            options );
        return this.each(function() {
            //best.plugin.ever
        });
    };
});
```

```

$.fn.bestPluginEver.defaults = {
    "color" : "blue",
    "title" : "my dashboard",
    "width" : "960px"
};

})( jQuery );

//通过重写原型属性设置全局默认属性
$.fn.bestPluginEver.width = "768px";
/*
 * 调用插件，传入一个选项参数，在单个实例中重写新的默认值
 */
$("div").bestPluginEver( {"width" : "960px" } );

```

代码片段 `public-settings-access.txt`

6. 为次要函数提供恰当的公有访问

这一最佳实践与上一小节中介绍的技术非常类似。二者关键的区别在于：前者仅仅向外界暴露了插件的设置，而这一小节中介绍的技术还向外界暴露了个别方法，允许外部对其进行修改和扩展。

在下面例子中，在插件体内定义了一个基础的 `sort` 函数。因为它被设计为可重写的，因此很容易为该插件定义一个不同的排序算法。



可从
Wrox.com
下载源代码

```

(function($) {
    $.fn.dataCruncher = function( options ) {
        var data = $.fn.dataCruncher.sort( data );
        return this.each(function() {
            //对数据进行某种处理
        });
    };

    $.fn.dataCruncher.sort = function(data) {
        for ( var j = 1 test = data.length; j < test; j++ ) {
            var key = data[j],
                i = j - 1;
            while ( i >= 0 && data[ i ] > key ) {
                data[ i + 1 ] = data[ i ];
                i = i - 1;
            }
            data[i + 1] = key;
        }
        return data;
    };
})( jQuery );

//提供一个新的排序函数
$.fn.dataCruncher.sort = function( data ) {

```



```

var len = data.length;
for ( var i = 0; i < len; i++ ) {
    var index = i;
    for ( k = i + 1; k < len; k++ ) {
        if ( data[k] < data[index] ) {
            index = k;
        }
    }
    var temp = data[ i ];
    data[ i ] = data[ index ];
    data[ index ] = temp;
}

```

代码片段 public-methods.txt

7. 保持私有函数的私有性

虽然可以将插件的部分功能暴露给外部以便进行增强，但这种方式会带来另外一个问题，即如果开发人员没有细心地标识哪些函数可以访问，哪些函数不允许访问的话，它可能会导致无法预测的结果。如果插件的某个功能非常重要，并且将其暴露给插件的使用者并不能提供任何实质性好处时，则最好保持该功能为私有。要使一个函数成为私有函数，只需要将该函数定义在插件函数的函数体内部、并位于返回对象之外即可。比如下面的代码：



可从
Wrox.com
下载源代码

```

(function($) {
    $.fn.dataCruncher = function(options) {
        //在插件之外不可访问
        function privateSort( data ){
            //私有方法
        }
        return this.each(function() {
            var data = privateSort( data );
        });
    };
});

```

代码片段 private-functions-private.txt

8. 支持 Metadata 插件

对于所有最初的建议，Metadata 插件是一个不像一开始推荐时那么重要的插件。由于 \$.data() 方法已经提供了类似的功能，因此 Metadata 插件本身已经被弃用。然而在某些情况下开发人员有可能需要使用较低版本的 jQuery——对旧代码进行重构，或者有时需要在无法更新 jQuery 核心库的环境中进行工作，因此学习 Metadata 插件还具有一定的价值。

Metadata 插件的功能是从一个 DOM 元素中抽取数据，并将其以一个对象的形式返回。它提供了方便的、基于标记的方法，用于定制插件。一个非常简单的例子就是：当需要抽取数据用于 CMS 之外的插件中时。比如允许 CMS 的作者通过单击一组复选框，然后通过 metadata 暴露这些选项以更新幻灯片选项。

很容易添加对 Metadata 插件的支持，因此在页面中引用插件的脚本时，可以很方便地添加 Metadata 插件的脚本。

下面代码示例演示了对 Metadata 插件的支持。它检测是否存在 \$.metadata，如果存在的话，它就将在 settings 和 options 对象中发现的任何数据合并，这些数据将传递给 CSS 以设置元素的样式。



可从
Wrox.com
下载源代码

```
<html>
<head>
<script src="http://code.jquery.com/jquery-1.7.1.js" ></script>
<script srcs="jquery.metadata.js"></script>
<script>
  (function($) {
    //插件定义
    var settings = {
      "background" : "#fe57a1",
      "color" : "#fff",
      "text-shadow" : "none"
    };
    $.fn.pinkify = function( options ) {
      return this.each(function() {
        var style = $.extend( settings, options ),
            $this = $( this );
        style = $.metadata ? $.extend( style, $this.metadata() ) :
            style;
        $this.css( style );
      });
    }
  })( jQuery );

  $( document ).ready(function() {
    $('.pinkify').pinkify();
  });
</script>
</head>
<body>
  <ul>
    <li class="pinkify"> Hot pink </li>
    <li class="pinkify { color : '#000' }">
      Black and Pink. Very hip.
    </li>
    <li class="pinkify { color : 'green' }">
      Not so sure about this one.
```



```

    </li>
  </ul>
</body>
</html>

```

代码片段 metadata.txt

12.2 学习和使用现有的插件模式

在具备了插件开发的基础知识后，你可能希望扩展插件开发的领域，进一步学习一些非常高级的插件开发技术。另外一些较新的资源可以参考 Addy Osmani 的文章 Essential jQuery Plugin Patterns(<http://coding.smashingmagazine.com/2011/10/11/essential-jquery-plugin-patterns/>)，以及相应的 Github 知识库(<https://github.com/addyosmani/jquery-plugin-patterns>)。在 Addy 的文章中，介绍了一些经过良好研究的插件开发的基本技术。这些内容既包含了一些简单的例子(比如下面的代码示例)——对这些例子中的技术你可能已经熟悉，还包含了一些支持模块模式——比如 RequireJS、Asynchronous Module Definition(AMD)和 CommonJS 的高级选项。

这些文章和相应的知识库是必不可少的学习材料，它既可以作为开发 jQuery 插件的简单起点，也是进一步学习的宝贵资源。下面的代码示例来自于 <https://github.com/addyosmani/jquery-plugin-patterns/blob/master/jquery.basic.pluginboilerplate.js>，它列出了一个 jQuery 插件的样板代码，其中包含了相应的注释。可以看到，模式中包含了大量的编程思想，并且每一行代码都有非常详细的说明。对于刚刚涉足 jQuery 插件开发工作的人而言，其中一些技术可能非常高级，但对于中级或高级 jQuery 插件开发者来说，这是一个极为重要的参考资源。



可从
Wrox.com
下载源代码

```

/*!
 * jQuery lightweight plugin boilerplate
 * Original author: @ajpiano
 * Further changes, comments: @addyosmani
 * Licensed under the MIT license
 */

// the semicolon before the function invocation is a safety
// net against concatenated scripts and/or other plugins
// that are not closed properly.
;(function ( $, window, document, undefined ) {

    //undefined is used here as the undefined global
    //variable in ECMAScript 3 and is mutable (i.e. it can
    //be changed by someone else). undefined isn't really
    //being passed in so we can ensure that its value is

```



```
//truly undefined. In ES5, undefined can no longer be
//modified.

//window and document are passed through as local
//variables rather than as globals, because this (slightly)
//quickens the resolution process and can be more
//efficiently minified (especially when both are
//regularly referenced in your plugin).

//Create the defaults once
var pluginName = 'defaultPluginName',
    defaults = {
        propertyName: "value"
    };

//The actual plugin constructor
function Plugin( element, options ) {
    this.element = element;

    //jQuery has an extend method that merges the
    //contents of two or more objects, storing the
    //result in the first object. The first object
    //is generally empty because we don't want to alter
    //the default options for future instances of the plugin
    this.options = $.extend( {}, defaults, options ) ;

    this._defaults = defaults;
    this._name = pluginName;

    this.init();
}

Plugin.prototype.init = function () {
    //Place initialization logic here
    //You already have access to the DOM element and
    //the options via the instance, e.g. this.element
    //and this.options
};

//A really lightweight plugin wrapper around the constructor,
//preventing against multiple instantiations
$.fn[pluginName] = function ( options ) {
    return this.each(function () {
        if (!$.data(this, 'plugin_' + pluginName)) {
            $.data(this, 'plugin_' + pluginName,
                new Plugin( this, options ));
        }
    });
};
```

```
})( jQuery, window, document );
```

代码片段 `jquery.basic.plugin-boilerplate.js`

从以上的例子可以看到，样板代码封装了很多本章所介绍的最佳实践，还添加了一些重要的增强，比如查询\$.data 中存储的插件，以避免重复实例化多个插件。

按照这样坚实的样板代码来开发插件，可以避免很多错误。另外，由于该项目在 Github 上，它将得益于庞大 jQuery 社区不断的审查和改进。

12.3 Widget Factory 概述

之前曾简要地介绍过 Widget Factory，它为 jQuery 插件开发工作提供了另外一种强大的开发方式。Widget Factory 虽然只是稍高级，但根据你正要编写的 jQuery 插件的风格，它提供了一些特性可以使开发人员从中受益。下面简要地列出了一些有用的特性，只要采用 Widget Factory 方法来开发 jQuery 插件，就可以自动获得这些特性。请注意，前面几个小节已经介绍过其中很多特性。这些特性都是 Widget Factory 方式自身就具有的。

- 对于同一对象，避免创建多个插件实例。
- 启用、禁用或销毁 widget 的方法。
- widget 是有状态的，因此销毁 widget 实例将使其返回 DOM 元素的初始状态。
- 创建名称空间和原型。
 - 从名称空间和名称生成一个伪选择器(pseudoselector)。
- 具有使用新选项覆盖插件默认设置的功能。
 - 暴露默认设置，以便设置插件的默认参数。
- 用于设置插件选项的方法和对插件选项变化做出响应的方法。
- 可以通过对象存储的数据访问插件实例。
- 自动的方法查找表。

虽然本章前面所看的关于 Widget Factory 的例子只是一行代码，但实际上使用 Widget Factory 来创建 jQuery 插件显然不止编写一行代码那么简单。

下面的代码展示了使用\$.widget 插件的最小代码。在\$.widget()方法中，传入一个名称空间作为该 widget 所属的名称空间，并传入一个对象字面量作为该 widget 的原型。该对象字面量应该包含一个 options 对象、一个_create 函数——当第一次调用 widget 时将自动运行该函数、一个_destroy 函数(如果使用 jQuery 1.8 版本则是 destroy 函数)——它的功能是销毁一个插件实例并清理 DOM、一个_setOption 函数——它的功能是响应插件 option 的改变，最后是开发人员定义的插件方法。

```
;(function ( $, window, document, undefined ) {
    $.widget( "best.widget" , {
        options: {
```



可从
Wrox.com
下载源代码

```

        hashtag: "#hotpink"
    },
    _create: function () {
        //创建
    },

    _destroy: function () {
        //销毁
    },
    _setOption: function( key, value ) {
        /* 在 jQuery UI 1.8 中, 必须从基 widget 中手工调用该 _setOption 方法 */
        $.Widget.prototype._setOption.apply( this, arguments );
        //在 jQuery UI 1.9 及更高版本中, 使用 _super 方法代替
        this._super( "_setOption", key, value );
    },
    pluginMethod : function(){
        //插件的功能
    }
});
})( jQuery, window, document );

```

代码片段 widget-factory.txt

12.4 插件开发示例

根据本章介绍的众多关于 jQuery 插件开发的知识, 我们可以采用多种方式来开发 jQuery 插件。条条大道通罗马。

开发人员应该根据所开发插件的目标, 选择开发模式和配置选项。

在下面的代码示例中, 插件接收一个 DOM 元素, 根据一些可用的 options, 在相同的位置上创建一个 HTML Canvas 元素, 该 canvas 元素包含了与目标元素相同的文本。在本例中, 该插件对页面上所有 H1 元素进行处理。

在下面这个示例插件之中, 使用了 Ben Alman 的全局调用和每次调用都可以重写的 Options(最佳 Options 模式)——这是 *Essential jQuery Plugin Patterns* 一文中的说法, 以便向外部暴露一个健壮的配置对象, 既可以在全局作用域中进行配置, 也可以在每次调用时进行配置。由于插件需要详细地设置它所创建的 Canvas 元素, 因此需要提供一个功能完备的配置。由于它可以对一个页面上的多个元素进行操作, 因此既允许重写默认设置, 也允许在每个调用时进行设置是非常有意义的。

虽然这是一个很小的插件, 但从中可以看到本章前面所讨论的多种技术。

- 在立即执行的函数表达式之前, 使用一个前导分号。

- 将\$、window、document 和 undefined 作为变量传入，以确保\$总是引用 jQuery、undefined 总是 undefined，还可以缩短查找 document 和 window 对象的时间。
- 通过一个 options 对象和 jQuery 方法\$.extend 进行配置。
- 返回 this，以支持链式方法调用。
- 将插件默认 options 暴露为全局配置。



可从
Wrox.com
下载源代码

```
;(function ( $, window, document, undefined ) {

$.fn.canvasizr = function ( options ) {
    options = $.extend( {}, $.fn.canvasizr.options, options );

    return this.each(function () {
        var $this = $( this ),
            pos = $this.position(),
            width = $this.outerWidth(),
            height = $this.outerHeight()
                + parseInt( $this.css( "margin-top" ) )
                + parseInt( $this.css( "margin-bottom" ) ),
            canvas = document.createElement( "canvas" ),
            $canvas = $( canvas ),
            ctx = canvas.getContext( "2d" );
        $.extend( ctx, options );
        canvas.width = width;
        canvas.height = height;
        ctx.fillRect( 0, 0, parseInt( width ), parseInt( height ) );
        if( options.border ){
            ctx.strokeRect( 0, 0, parseInt( width ), parseInt( height ) );
        }
        ctx.fillStyle = ctx.textColor;
        ctx.fillText( $this.text(), 8, parseInt( height )/2 );
        $canvas.css({
            "position" : "absolute",
            "left" : pos.left + "px",
            "top" : pos.top + "px",
            "z-index":1
        });
        $( "body" ).append( $canvas );
    });
};

$.fn.canvasizr.options = {
    textColor : "#ffffff",
    fillColor: "#ff0000",
    strokeStyle : "#000",
    border: false,
    font : "20px sans-serif",
    lineCap : "butt",
    lineJoin : "miter",

```

```

        lineWidth : 1,
        miterLimit : 10,
        shadowBlur : 0,
        shadowColor : "rgba(0, 0, 0, 0)",
        shadowOffsetX : 0,
        shadowOffsetY : 0,
        textAlign : "start",
        textBaseline : "alphabetic"
    };

})( jQuery, window, document );

```

代码片段 canvasizr.txt

下面的代码示例说明了如何使用该插件。一开始，定义了一个默认的样式，用于该插件的所有实例。随后，两次调用了该插件，第一次在#special 元素上调用该插件，第二次则是在#first、#second 和#third 这三个元素的集合上调用该插件。在每次调用该插件时，声明了不同的 options 参数，从而产生不同的结果。该示例代码的输出效果如图 12-1 所示。



可从
Wrox.com
下载源代码

```

$.fn.canvasizr.options.fillStyle = "#fe57a1";
$( "#special" ).canvasizr({
    font : "30px Consolas, 'Lucida Console', monospace"
});
$( "#first, #second, #third" ).canvasizr({
    textColor : "#ffffff",
    fillStyle: "#ff0000",
    font : "40px sans-serif",
    border:true
});

```

代码片段 canvasizr-called.txt

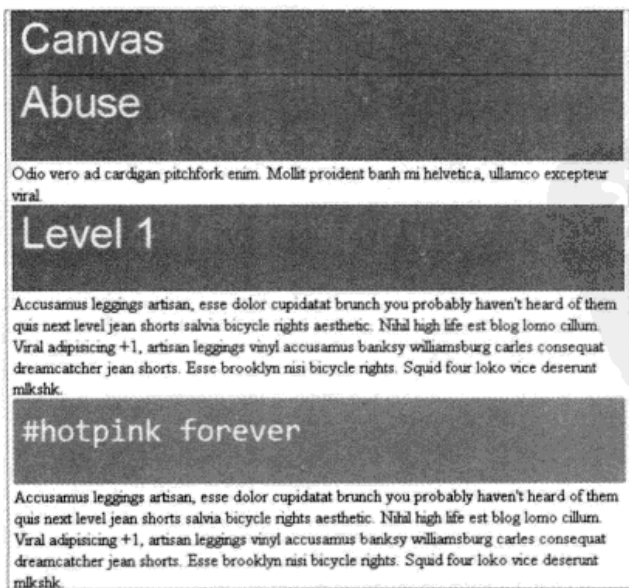


图 12-1

12.5 小结

经过本章的学习，你应该已经对 jQuery 插件开发具备了坚固的基础知识。通过对这些基础知识的透彻理解以及对一些高级技术的实践，你已经可以创建出强大的、可配置的 jQuery 插件，并且使自己创建的 jQuery 插件能够与其他代码很好地协同工作。本章介绍了如何扩展 jQuery、如何暴露多种配置 options 和增强 options。本章还介绍了如何提供通用特性、如何实现最好的代码风格以及如何保持插件的体积最小化，从而使你编写的 jQuery 插件能够很好地融入 jQuery 的生态系统之中。

本章还介绍了一些宝贵的学习资源，供你进一步研究 jQuery 插件的开发技术。



第 13 章

使用 jQuery Deferred 对象进行高级异步编程

本章内容

- Promises/A Proposal
- jQuery Deferred 对象

本章将介绍 jQuery 的 Deferred 对象。\$.Deferred 是在 jQuery 1.5 版本中引入的，它是一个可链式调用的工具对象，它为回调函数的处理提供了精细的控制。例如，在以往的 \$.ajax 调用中仅仅只有一个 success 方法，如果使用 Deferred 对象的话，可以定义多个回调函数，并且可以将多个回调函数准确而灵活地组合在一起。

本章还将介绍 CommonJS Promises/A proposal，它是 \$.Deferred 的设计基础。在介绍了 CommonJS Promises/A proposal 的基础知识之后，将详细介绍 jQuery 中 \$.Deferred 的实现。

充分利用 \$.Deferred 可以消除一些令人不愉快的代码模式。庞大的 success 代码块、复杂的应用程序逻辑都已经成为过去。在异步请求中棘手的函数调用的时机问题、在一个函数的末尾调用另外一个函数，从而造成函数调用难以维护的问题，现在都可以抛诸脑后了，\$.Deferred 使这一切问题都迎刃而解。\$.Deferred 将所有异步调用的逻辑提升了一个层次，使之具有一个清晰的结构，使 jQuery 中异步调用的代码更灵活、具有更好的可维护性。\$.Deferred 使编写异步调用代码就像 jQuery 的日常用法一样，成为开发人员乐于使用的一种模式。

13.1 \$.Deferred 基础

\$.Deferred 构建在已有的工作和概念之上，它来源于 jQuery 项目之外。要完全理解 \$.Deferred 的特性，直接查看它的源代码是非常有用的。这一小节将简要地介绍一下 Deferred 模式的来源，这将有助于理解 jQuery 中 \$.Deferred 的具体实现。

13.1.1 Promise

根据基本的定义，一个 promise 就是一个行为结果的代理，该行为将在将来一个未指定的时间点发生。

虽然 promise 的概念对于 jQuery 相对较新，但在软件设计模式中，promise 已经存在很长时间了。promise 的名称和概念，可以追溯到 Daniel P. Friedman 和 David Wise 在 1976 年写的一篇文章。尽管这篇论文在语言的实现上学院风格重于实践，但在 Java、Scala、Ruby、Perl、Python 和 Common Lisp 中，都存在 promise 库的实现。就近期而言，从 dojo 框架的 0.9 版本开始，已经包含了 promise 模式的一个主要实现。

由于 promise 模式很容易实现异步编程，特别适合于 Web 程序设计，因此在 jQuery 中不可避免地需要引入 Promises/A Proposal。

13.1.2 Promises/A Proposal

jQuery 中 \$.Deferred 的具体实现，是基于 CommonJS 的 Promises/A proposal。

CommonJS 是一个以 JavaScript 互用性为中心的工作组，它致力于发展健壮的 JavaScript 生态系统，使开发人员获得其他语言中所具有类库支持，比如 Python、Ruby 和 Java。该工作组主要关注于模块的问题，这些模块是由 Common JS Modules API 直接声明的。但他们也对其他一些语言特性展开工作，其中之一就是 promise。在维基百科上，CommonJS 列出了 Promises 模式的几个建议。Promises/A Proposal 来自于 Kris Zyp。下面是 Promises/A Proposal 的定义：

一个 promise 代表了一次完成某个操作时最终的返回值。一个 promise 可以是以下三种状态之一：unfulfilled、fulfilled 和 failed。promise 的状态只能从 unfulfilled 改变为 fulfilled，或者从 unfulfilled 改变为 failed。一旦一个 promise 的状态为 fulfilled 或者 failed，则 promise 的值就绝不允许改变。promise 的不可变性对于避免副作用是非常重要的，这一特性可以避免 listener 的行为对 promise 造成无法预料的改变，并允许将 promise 传递给其他函数，而不会对调用者造成影响。与之类似，可以将原始的 promise 对象传递给函数，无须担心被调用函数会修改调用者的变量。

一个 promise 被定义为一个对象，该对象的 then 属性的值是一个函数。

<http://wiki.commonjs.org/wiki/Promises/A>

下面的代码示例简要地展示了 Promises/A Proposal 中的一些概念。

假如 ajaxPromise 返回一个 promise 对象,那么该 promise 对象可以是 unfulfilled、fulfilled 或者 failed 这三种状态之一。下面的例子描述了 promise 对象常见的用法:



```
var promise = ajaxPromise() {
    //XHR 请求
    //根据请求结果,将状态设置为 fulfilled、unfulfilled 或者 failed
    return state
};

function unfulfilled(){
    //处理无效请求
}
function fulfilled() {
    //处理成功的请求
}
function failed(){
    //错误
}

promise.then(
    unfulfilled,
    fulfilled,
    failed
);
```

代码片段 pseudocode.txt

ajaxPromise 启动后,将通过一个 XMLHttpRequest 请求获取一些数据,当该请求结束时,它返回一个 promise 对象,该 promise 对象处于 unfulfilled、fulfilled 或者 failed 这三种状态之一。在上面的代码中,设计了三个函数,分别用于处理这三种状态。then()方法根据 promise 对象的状态,将其路由到其中一个函数进行处理。请注意,这些逻辑都没有绑定于 ajaxPromise 自身,它们都是在函数外进行处理的,这使 ajaxPromise 更为通用,因为应用程序的逻辑没有直接绑定该方法。

经过上面的介绍,我们已经了解了 Promises/A Proposal 概念的历史渊源,在下面的小节中,将介绍 jQuery 中 Deferred 对象实现的强大功能。

13.2 jQuery 中的 Deferred 对象

jQuery 中实现的 Deferred 对象,在 promise 对象上增加了一些基本前提和一些有用的增强。更为重要的是,Deferred 对象直接集成到了 jQuery 库的核心特性 \$.ajax 中,因此 Deferred 对象打开了 jQuery 异步编程的大门,解除了曾经困扰开发人员的一些难题。

按照 jQuery 的惯例,Deferred 对象在 jQuery 中的实现必须方便地支持方法的链式调用,

这一点很重要。另外还应该以一种用户友好的、可读的风格来编写。

下面将介绍 jQuery 中 Deferred 对象的基本实现，并介绍一些较为高级的有用特性。

\$.when、Deferred.done 和 Deferred.fail

在使用 Deferred 对象时，会发现在 jQuery 的新方法 \$.when 中包含了很多执行流。\$.when 既可以接收一个或多个 Deferred 对象(这在 \$.ajax 调用中非常重要)作为参数，也可以接收一个普通对象作为参数。

如果传递给 \$.when 的是一个 Deferred 对象，则该方法将返回这个 Deferred 对象的 promise 对象。在后面的小节中将更详细地介绍如何管理 jQuery 的 promise，在 \$.when 方法之后可以链接其他方法并安排回调函数的结构。

先介绍其中两个方法：Deferred.done 和 Deferred.fail。

Deferred.done 接收一个函数或一个函数的数组作为参数，当 promise 对象成功 resolved 时将触发该函数的执行。

Deferred.fail 也接收一个函数或一个函数的数组作为参数，当 promise 对象被 rejected 时将触发该函数的执行。

请注意，如果传递给 \$.when 的参数不是一个 Deferred 对象，则 \$.when 将把该参数视为一个 resolved 的 Deferred 对象。如果没有其他 Deferred 对象传入 \$.when，则触发 Deferred.done 的执行。

下面代码使用 \$.when、Deferred.done 和 Deferred.fail，简明扼要地列出了 jQuery Deferred 对象的一个实例。

在下面的例子中定义了三个函数。fib 函数执行一些算术运算并将计算结果显示为一个无序列表。此外还定义了两个函数 success 和 failure。这两个函数都很简单，它们的功能是根据第一个函数的结果修改页面中一个 <h1> 元素的文本。这段代码最有趣的功能，是它将根据 promise 对象是 resolved 还是 rejected 来触发 success 函数或 failure 函数的调用。



可从
Wrox.com
下载源代码

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
  <div id="main">
    <h1></h1>
    <ul id="numbers">
    </ul>
  </div>
  <script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
  <script>
    function fib() {
      var int1 = 0,
        int2 = 1,
```

```

    int3,
    sequence = "<li>0</li><li>1</li>";
    for ( var i = 3; i <= 100; i++ ) {
        int3 = int1 + int2;
        int1 = int2;
        int2 = int3;
        sequence += "<li>" + int3 + "</li>"
    }
    $( "#numbers" ).append( sequence ).show( 500 );
}
function success() {
    $( "h1" ).text( "Fibonacci!" );
}
function failure() {
    $( "h1" ).text( "No numbers?" );
}
$(function(){
    $.when( fib() )
        .done( success )
        .fail( failure );
});
</script>
</body>
</html>

```

代码片段 basic-deferred.txt

如果你已经认真阅读过前面的内容，应该还记得：如果将一个函数传递给\$.when，并且该函数不是一个 Deferred 对象，那么\$.when 方法将把该函数视为一个 resolved 的 Deferred 对象，并调用 Deferred.done 方法。在上面这个例子中，Deferred.fail 方法永远都不会触发。

因此尽管该例子是有效的，并且基本能按照预期效果工作，它演示了使用\$.when 的基本模式，但并没有完全展示出\$.when 的特点。在下面的代码示例中，重新定义了 fib 函数以充分利用 Deferred 对象。现在 fib 函数将返回一个 Deferred.promise 对象。之前 fib 函数中的逻辑将作为参数传入\$.Deferred()。

最值得注意的要点是：promise 对象的结果将作为\$.show 方法的回调函数。在下一小节中还将更深入地讨论这一重要模式。



可从
Wrox.com
下载源代码

```

<!doctype html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
    <div id="main">
        <h1></h1>
        <ul id="numbers">

```

```

        </ul>
    </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script>
function fib() {
    return $.Deferred(function() {
        var int1 = 0,
            int2 = 1,
            int3, sequence = "<li>0</li><li>1</li>";
        for (var i = 3; i <= 100; i++) {
            int3 = int1 + int2;
            int1 = int2;
            int2 = int3;
            sequence += "<li>" + int3 + "</li>";
        }

        $("#numbers")
            .append(sequence)
            .show(1000, this.resolve);
    }).promise();
}

function success() {
    $( "h1" ).text( "Fibonacci!" );
}
function failure() {
    $( "h1" ).text( "No numbers?" );
}
$(function(){
    $.when( fib() )
        .done( success )
        .fail( failure );
});
</script>
</body>
</html>

```

代码片段 `proper-deferred.txt`

虽然这是一个简单的例子，但它说明了 Deferred 模式的威力。在 jQuery 中，回调函数通常都是造成程序结构紧耦合的根源。通常将回调函数再定义为匿名函数，很少关心对回调函数的机制进行抽象。

下面是一个动画的示例，它清楚地描述了 Deferred 如何解除程序逻辑的耦合，从而更好地支持代码重用和一个总体上更加清晰的代码架构。如果没有 Deferred 对象，那么动画效果的回调逻辑将直接绑定于动画调用，并作为一个可选的 `complete` 参数。`complete` 参数被定义为一个函数，当动画结束时将触发该函数的执行。在下面的第一段代码中，演示了传统方式控制动画执行的情况：



```

<!doctype html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
  <div id="main">
    <div id="statusbar" style="display:none">
    </div>
  </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script>
function updateStatus() {
  var $update = $("<ul />"),
  $statusbar = $("#statusbar"),
  html = []; //缓存文本
  for ( var i = 0, test = 20; i < test; i++ ) {
    html.push( "<li>status update</li>");
  }
  html = html.join("\n"); //将缓存文本转换为字符串
  $update.append(html);
  $statusbar.append($update);
  $statusbar.slideDown(5000, function() {
    console.log("animation is done! On to the next operation");
  });
}

$(function(){
  updateStatus();
});
</script>
</body>
</html>

```

代码片段 tight-callbacks.txt

对于上面的例子，虽然在 `complete` 参数函数中只有一个简单的 `console.log` 输出，但是更加复杂的应用程序数据通常都被包装在与之类似的 `complete` 回调函数中。

如果在 `updateStatus` 方法中只使用匿名回调函数，那么上面的代码是可以接受的。但如果需要与其他函数匹配，就得采用其他的办法。就代码重用性而言，将深藏在回调函数中的逻辑从 `updateStatus` 内部抽取出来，可以获得更高的灵活性。

要解决这一问题，另外一种办法是为 `updateStatus` 方法添加一个参数，并传入一个回调函数作为该参数的值。但使用 `Deferred` 并返回一个 `resolved` 的 `promise` 对象则更加灵活。下面的代码示例说明了如何重写 `updateStatus` 方法以充分利用 `Deferred` 对象的特性：



```
function updateStatus() {
    return $.Deferred(function() {
        var $update = $("<ul />"),
            $statusbar = $("#statusbar"),
            html = []; //缓存文本
        for ( var i = 0, test = 20; i < test; i++ ) {
            html.push( "<li>status update</li>" );
        }
        html = html.join("\n"); //将缓存文本转换为字符串
        $update.append(html);
        $statusbar.append($update);
        $statusbar.slideDown(1000, this.resolve);
    }).promise()
}
```

代码片段 *decoupling.txt*

上面的例子只是 Deferred 对象的一个简单应用，在采用 Deferred 对象之后，现在的 updateStatus 方法已经非常灵活。除了可以绑定到单个回调函数之外，现在还可以将 updateStatus 方法应用在多个不同的执行流中。下面的代码块演示了充分利用 Deferred 对象之后，updateStatus 方法的几种使用方式。

下面的例子包含了三个回调函数。第一个回调函数等价于前面第一个例子中作为回调函数传入的匿名函数。第二个函数 alternativeCallback 表示在动画完成之后的另外一条不同的路径。如果在第二个页面上运行相同的动画，则需要更新另外一段标记代码。第三个函数 happyBirthday 代表了需要在特定情况下调用的一个函数，此时需要更新特定的 UI，比如一个 HAPPY BIRTHDAY 的消息提示。在这三个函数的定义之后，使用了三个 \$.when 方法。第一个 \$.when 方法通过一个回调函数重新实现了原先的例子。第二个 \$.when 显示了另外一个回调函数。第三个 \$.when 方法演示了如何执行多个回调函数。



```
function callback(){
    console.log("The animation is done. On to the next operation");
}
function alternativeCallback(){
    console.log("The animation is done. Let's follow a different path.");
}
function specialCase(){
    console.log("This is a special case. Let's do a special UI update.");
}

$.when( updateStatus() )
    .done( callback );

//另外一个回调函数
$.when( updateStatus() )
    .done( alternativeCallback );
```

```
//多个回调函数
$.when( updateStatus() )
    .done(
        [ callback,
          specialCase ]
    );
```

代码片段 `decoupled-callbacks.txt`

可以看到，上面的代码很好地将回调函数的逻辑从 `updateStatus()` 方法体内解耦出来。这种方式可以创建可重用性更好的代码，也为项目中的其他开发人员提供了一个非常清晰的模式。其他开发人员不必再阅读 `updateStatus()` 方法的代码，就可以知道应用程序的执行流。它们用几乎纯英语的方式表达了出来。

用 `Deferred.then` 语法包装 `Deferred.fail` 和 `Deferred.done`

`$.get` 和 `$.post` 是 `$.ajax` 方法的别名，与之类似，jQuery 为核心 `Deferred` 对象提供了一个方便使用的别名：`Deferred.then`。`Deferred.then` 接收两个参数，第一个参数用于处理 `resolved promise` 的情形；第二个参数用于处理 `rejected promise` 的情形。与它所映射的函数 `Deferred.done` 和 `Deferred.fail` 类似，`Deferred.then` 既可以接收单个函数作为参数，也可以接收函数的数组作为参数。

下面的代码示例使用 `Deferred.then` 方法，重写了本章前面最初的 `Deferred` 对象的示例。



```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
  <div id="main">
    <h1></h1>
    <ul id="numbers">
    </ul>
  </div>
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script>
function fib() {
  return $.Deferred(function() {
    var int1 = 0,
    int2 = 1,
    int3, sequence = "<li>0</li><li>1</li>";
    for (var i = 3; i <= 100; i++) {
      int3 = int1 + int2;
      int1 = int2;
      int2 = int3;
```



```

        sequence += "<li>" + int3 + "</li>";
    }
    $("#numbers")
        .append(sequence)
        .show(1000, this.resolve);
    }).promise();
}

function success() {
    $( "h1" ).text( "Fibonacci!" );
}
function failure() {
    $( "h1" ).text( "No numbers?" );
}
$(function(){
    $.when( fib() )
        .then( success, failure );
});
</script>
</body>
</html>

```

代码片段 deferred-then.txt

由于 Deferred 方法是可链接的，因此 Deferred.then 方法可以与 Deferred.done、Deferred.fail 甚至另外的 Deferred.then 方法结合在一起。

在\$.ajax 中使用 Deferred 对象

前面曾经介绍过，\$.ajax 是一个 Deferred 对象，它的 success、error 和 complete 回调函数分别与 Deferred.done、Deferred.fail 和 Deferred.always 类似。下面的代码列出了一个典型的 \$.get 请求：



```

$.get("/status/json/",
    function( data ) {
        var $update = $( "<ul />" ),
            $statusbar = $( "#statusbar" ),
            html = "",
            statusMessages = data.statusMessages;
        for ( var i = 0, test = statusMessages.length; i < test; i++ ) {
            html += "<li>" + status[i] + "</li>";
        }
        $update.append(html);
        $statusbar.append($update);
        $statusbar.slideDown(1000);
    }
);

```

代码片段 non-deferred-ajax.txt

接下来的代码块使用 Deferred 对象实现了相同的 Ajax 请求。值得注意的是, 传入 `updateStatus()` 方法的参数, 就是在典型 Ajax 请求中传入回调函数的参数, 二者是相同的。该参数包含了 `data`、一个指示该 Ajax 请求状态的字符串、一个 jQuery XMLHttpRequest(jqXHR) 对象。另外, 在 jQuery 1.5 中利用 Deferred 对象重写了 `$.ajax`, jqXHR 对象是标准 XMLHttpRequest 对象的一个增强版本。它完全具备所有的 Deferred 方法, 因此对于标准 Deferred 对象的任何方法, 在所有 `$.ajax` 请求中都可以按照类似的方式使用。



可从
Wrox.com
下载源代码

```
function getStatus() {
    return $.ajax({
        url : "/status/json/",
        dataType : "json"
    })
}

function updateStatus( data ) {
    var $update = $( "<ul />" ),
    $statusbar = $( "#statusbar" ),
    html = "",
    statusMessages = data.statusMessages;
    for ( var i = 0, test = statusMessages.length; i < test; i++ ) {
        html += "<li>" + statusMessages[i] + "</li>";
    }
    $update.append( html );
    $statusbar.append( $update );
    $statusbar.slideDown( 1000 );
}

$.when( getStatus() )
    .done( updateStatus );
```

代码片段 deferred-ajax.txt

从表面上看, 上面的代码对传统的回调函数方法似乎没有太大的改进。与传统 jQuery Ajax 请求中的回调函数相比, 新代码更长、自由度似乎更小。但是值得注意的是, 回调函数的逻辑已经从 Ajax 请求中解耦出来。这样, `getStatus` 函数立即就变得更加灵活。此外, 与其他任何 Deferred 对象一样, 可以将多个 `$.ajax` 请求传入 `$.when` 方法, 也可以将多个回调函数链接起来以响应 promise 对象状态的变化, 因此它极大地扩展了处理 Ajax 请求的各种可能性。下面的例子虽然有点夸张, 但有助于你理解。



可从
Wrox.com
下载源代码

```
$.when(
    $.post("/echo/json",
        function() {
            console.log("1")
        })
)
```

```
    ),
    $.post("/echo/json",
        function() {
            console.log("2")
        }
    ),
    $.post("/echo/json",
        function() {
            console.log("3")
        }
    )
).then([
    function() {
        console.log("4")
    },
    function() {
        console.log("5")
    },
    function() {
        console.log("6")
    }
]).then([
    function() {
        console.log("7")
    },
    function() {
        console.log("8")
    },
    function() {
        console.log("9")
    }
]).then([
    function() {
        console.log("10")
    },
    function() {
        console.log("11")
    },
    function() {
        console.log("12")
    }
]).done(
    function() {
        console.log("Electric Company!")
    }
);
```

代码片段 multiple-callbacks.txt

使用 Deferred.always 执行一个函数，无论 promise 是否已解决

很多时候，无论 promise 是否已经解决都需要调用一个函数。这与 complete 回调函数类似，无论一个 \$.ajax 请求是否已经解决都会触发 complete 回调函数的执行。对于 Deferred 对象来说，使用 Deferred.always 方法来执行类似的功能。与其他 Deferred 方法类似，Deferred.always 接收单个函数或者一个函数数组作为参数，无论一个 promise 是 resolved 还是 rejected 都会触发 Deferred.always 的执行。下面的代码给出了一个简单的例子，它的功能是在一个邮件应用程序中修改“最近更新”标识。



```
$.when(
    $.ajax( "/get/mail/" )
).done(
    newMessages,
    updateMessageList,
    updateUnreadIndicator
).fail(
    noMessages
).always(
    function() {
        var date = new Date();
        $( "#lastUpdated" ).html( "<strong>Folder Updated</strong>: "
            + date.toDateString()
            + " at "
            + date.toTimeString()
        );
    }
)
```

代码片段 *always.txt*

使用 Deferred.pipe 链接和过滤 Deferred

在 jQuery 1.6 中引入了 Deferred.pipe，它提供了一个过滤和进一步链接 Deferred 对象的方法。Deferred.pipe 方法常见的用途是利用它的链接功能，链接 Deferred 对象以创建一个动画队列。下面的代码是一个简单的例子，它将动画链接起来，以方便对应用程序中的动画组件进行管理。一旦动画队列完成，promise 将自动 resolved，并触发 done 方法的执行。



```
function buildpage() {
    return $.Deferred(function( dfd ) {
        dfd.pipe(function() {
            return $( 'header' ).fadeIn();
        })
        .pipe(function() {
```

```

        return $( '#main' ).fadeIn();
    })
    .pipe(function() {
        return $( 'footer' ).fadeIn();
    })
    ).resolve();
}
$.when( buildpage() )
    .done(function() {
        console.log('done')
    })
    );

```

代码片段 deferred-pipe-animation.txt

除了方便地链接功能之外，Deferred.pipe 还具有其他功能。它提供了一个非常简便的方法，可以根据二级条件来过滤 Deferred。Deferred.pipe 接收三个参数：doneFilter、failFilter 和 progressFilter。progress 的概念将在后面的小节中介绍。就目前而言，我们只关注过滤标准的 Deferred 对象。

下面的代码演示了一个根据 Ajax 请求返回的数据来过滤一个 \$.ajax 响应的例子。该示例扩展了前面介绍的基础邮件更新应用程序。在下面的例子中，假定邮件服务总是返回一个范围在 200 以内的成功的 HTTP 响应，即便在没有邮件时也是如此。因此，\$.ajax 的 success 函数总是被调用。通过使用 Deferred.pipe，可以检查邮件服务提供的数据，并测试在 data 对象中是否有 messages。如果有 messages，则认为响应是有效的，成功的响应和数据将传递给 Deferred.done 方法。如果没有 messages，则使用 Deferred.reject() 方法拒绝该 promise。这将触发 Deferred.fail 方法的执行。



可从
Wrox.com
下载源代码

```

$.when(
    $.ajax( "/get/mail/" )
).pipe(
    function( data ) {
        if ( data.messages.length > 0 ) {
            return data ;
        } else {
            return $.Deferred().reject();
        }
    }
).done(
    newMessages,
    updateMessageList,
    updateUnreadIndicator
).fail(
    noMessages
).always(
    function() {

```

```

    var date = new Date();
    $("#lastUpdated").html("<strong>Folder Updated</strong>: "
        + date.toDateString()
        + " at "
        + date.toTimeString()
    );
}
);

```

代码片段 *advanced-pipe.txt*

完成和拒绝 promise

从本章前面的例子可以看到,有时我们需要手工地完成或拒绝一个 promise。在前面的例子中使用了两个最常见的方法来实现该功能: `Deferred.resolve` 方法和 `Deferred.reject` 方法。但前面的例子没有介绍可选的 `args` 参数。根据需要,这两个方法都可以接收一个可选的参数,根据 promise 解决的情况,该参数将传递给对应的 `Deferred.done` 或 `Deferred.fail` 方法。

下面的例子简单地演示了 `Deferred.resolve` 方法和 `Deferred.reject` 方法中可选参数的应用。让我们继续改进前面邮件服务的例子,在下面的代码中,创建了一个更新日期字符串以反映邮件更新的时间。该字符串将传递给 `Deferred.fail` 方法和 `Deferred.done` 方法。另外,还将 `messages` 的数量传递给 `Deferred.done` 方法,以更新 `#message` 元素显示当前接收到的新 message 的数量。



可从
Wrox.com
下载源代码

```

function newMessages( obj ) {
    $( "#message" ).text("you updated at "
        + obj.date
        + " and have "
        + obj.number
        + " new messages"
    )
}
function noMessages( obj ) {
    $( "#message" ).text("you updated at "
        + obj.date
        + " and have no new messages"
    )
}
$.when(
    $.ajax( "/get/mail/" )
).pipe(
    function( data ) {
        var date = new Date();
        date = date.toDateString() + " at " + date.toTimeString();
        if ( data.messages.length > 0 ) {

```



```

        return $.Deferred().resolve({
            date: date,
            number: data.messages.length
        });
    } else {
        return $.Deferred().reject({
            date: date
        });
    }
}
).done(
    newMessages
).fail(
    noMessages
);

```

代码片段 *resolving-and-rejecting.txt*

另外，还有两个与之相应的方法 `Deferred.rejectWith()` 和 `Deferred.resolveWith()`，可以在完成或拒绝一个 `Deferred` 对象时，将传入的第一个参数作为特定的 `this` 上下文。

使用 `Deferred.progress` 和 `Deferred.notify` 跟踪进度

在 jQuery 1.7 中增加了 `Deferred.progress` 和 `Deferred.notify` 方法，结合这两个方法可以在长时间运行的函数中根据进度进行更新。

这两个方法的基本使用很简单。只需要在应用程序的不同执行点，设置一些专门设计用于在应用程序或长时间运行方法的特定执行点上触发的 `notify` 方法。`Deferred.notify` 方法接收一个可选的参数，该参数将传递给 `Deferred.progress` 回调函数。相应的 `Deferred.notifyWith` 方法还可以接收一个 `context` 参数，该参数用于指定 `Deferred.progress` 回调函数的 `this` 上下文。`Deferred.progress` 回调函数将被添加到调用链中。回调函数应该设计为既可以处理 `Deferred.notify` 发送的消息，也可以处理 `Deferred.notifyWith` 发送的消息。

下面是一个简单的例子，`longRunning` 函数是一个 `Deferred` 对象，它使用 `setTimeout` 延迟 5 000 毫秒后完成该 `promise`。在计时器启动之前，先发出了一个通知字符串。`progress` 函数将接收该 `notification` 参数，并将更新文本插入到 DOM 中。在该过程完成后，再次用 `notify` 方法发出一个通知表示该过程已经完成，然后手工完成该 `promise`。

理论上，更新 `#notifier` 文本的工作可以用 `Deferred.done` 回调函数进行处理，让 `progress` 回调函数仅处理开始时发出的通知。但在该过程的开始和结束都采用 `Deferred.notify` 发送通知显得更为对称。另外，这样处理可以使重复的代码最少，因此用 `progress` 回调函数来处理所有的通知是最好的。



可从
Wrox.com
下载源代码

```

var longRunning = function() {
    return $.Deferred(function(dfd) {
        dfd.notify( "operation started" );
    });
}

```

```

    var callback = function() {
        dfd.notify( "operation finished" );
        dfd.resolve();
    }
    setTimeout( callback, 5000 );
    }).promise();
}
longRunning().progress(
    function( notification ) {
        $( "#notifier" ).text( notification ).fadeIn( 500 );
    }).done(function() {
        $( "#notifier" ).css({
            "color" : "green",
            "font-weight" : "bold"
        })
    })
});

```

代码片段 deferred-progress.txt

使用 Deferred.state 检查 Deferred 对象的状态

如果开发人员需要检查 Deferred 对象的解决状态(当深入使用 Deferred 对象后一定需要), 可以使用 Deferred.state 进行检查。

Deferred.state 是一个简单的工具函数, 它返回一个表示 Deferred 对象当前状态的字符串。它可以返回三个值: pending、resolved 和 rejected。

表 13-1 列出了这三种状态的定义。

表 13-1 Deferred 的可能状态

状 态	定 义
pending	Deferred 对象处于未完成状态
resolved	Deferred 对象处于完成状态, 这意味着已经调用了 Deferred.resolve() 方法并且 Deferred 对象的 done 回调函数已经触发
rejected	Deferred 对象处于拒绝状态, 这意味着已经调用了 Deferred.reject() 方法并且 Deferred 对象的 fail 回调函数已经触发

在上一小节中介绍了一个长时间运行的 longRunning 函数的例子, 下面的例子在前例的基础上添加了几个对 Deferred.state 的调用, 说明了如何使用 Deferred.state。



可从
Wrox.com
下载源代码

```

var longRunning = function() {
    return $.Deferred(function( dfd ) {
        dfd.notify( "operation started" );
        console.log( dfd.state );
    });
}

```

```
var callback = function() {
    dfd.notify( "operation finished" );
    dfd.resolve();
}
setTimeout( callback, 5000 );
}).promise();
}

longRunning().progress(
    function( notification ) {
        console.log( dfd.state );
        $( "#notifier" ).text( notification ).fadeIn(500);
    }).done(function() {
        console.log( dfd.state );
        $( "#notifier" ).css({
            "color" : "green",
            "font-weight" : "bold"
        })
    });
```

代码片段 deferred-state.txt

13.3 小结

经过本章的学习，你应该对 jQuery Deferred 对象的强大功能以及它的常用方法有了一个完整的了解，

本章介绍了一些使用 Deferred 对象的例子，在这些例子中，通过 Deferred 对象将应用程序的逻辑从函数体中抽取出来，放在 Deferred 对象提供的 Deferred.when、Deferred.then 和 Deferred.done 等方法中，这些方法为程序的代码提供了具有更好可维护性，也更加灵活的结构。

本章还介绍了如何以手工方式创建、完成和拒绝一个 promise，这是发挥 Deferred 对象威力的关键。

本章还介绍了一些 Deferred 对象的功能，比如在 Deferred 对象的处理过程中，使用 Deferred.notify 和 Deferred.progress 方法发送和处理进度通知。

在学习和掌握了 Deferred 对象之后，你可以重新审视现在和之前的代码库，充分利用 Deferred 所提供的功能和结构对旧式代码进行重构。Deferred 对象是 jQuery 1.7 中新增加的最为重要的特性之一，它为 jQuery 程序带来了更清晰的结构和更高的灵活性。

第 14 章

使用 QUnit 进行单元测试

本章内容

- 单元测试简介
- QUnit 基础
- 异步测试
- 综合示例

本章将介绍单元测试的优势，介绍 jQuery 项目自身创建和使用的特殊的单元测试框架：QUnit。本章将介绍如何编写测试、如何将项目集成到 QUnit 中，以提高代码的质量、可维护性和对代码的整体信心。

14.1 单元测试简介

如果你之前受过其他程序设计语言的训练，现在转入 jQuery 和 JavaScript 开发，那么你对单元测试的概念应该很熟悉。如果是这种情况，那么你已经是 Web 前端工程师中少数已经使用过单元测试的人，可以略过介绍单元测试的内容，直接跳到后面专门介绍 QUnit 进行单元测试的章节。如果你还没有单元测试的基础知识，那么请阅读前面内容，学习单元测试——一种以结构化的自动测试框架为中心的开发方式——所带来的好处。

值得一提的是，对于前端工程师而言这是一个取得领先优势的机会。根据 DailyJS JavaScript Developer Survey 2011(<http://dailyjs.com/2011/12/15/javascript-survey-results/>)的结果显示，仅有 42% 的 JavaScript 工程师使用单元测试对代码进行测试。虽然自 2010 年以来该数字已经在增长，但这一结果依然说明在普通 JavaScript 项目中使用单元测试工具，与在其他程序设计语言项目中使用单元测试工具之间还存在着较大的差异。

单元测试的优势非常明显，因此对于 JavaScript 开发人员来说，值得付出一定的努力，

学习和使用 QUnit 单元测试技术。

14.2 什么是单元测试

单元测试是一种软件测试实践，它对单独的代码块(称为 Unit，即单元)进行测试以判断代码单元对于具体任务的可靠功能。代码单元应该尽可能地小、只具有一个或少量输入，并且具有单个输出结果。

单元测试通常都是自动化的，它要么作为 build 过程的一部分，要么作为签入的一个 pre-commit hook，或者软件生命周期中其他阶段的一部分。每一个测试都应该是 pass/fail 这两种结果其中之一。

JavaScript 开发人员对另外一种风格的测试更加熟悉——功能测试，单元测试是功能测试至关重要的搭档。绝大多数开发人员将功能测试视为 QA，或者更正式地称之为验收测试，在软件开发完成后，它从最终用户的角度来测试网站或应用程序。单元测试与功能测试模型刚好相反。单元测试是与代码一起编写的，因此最重要的区别在于：单元测试的编写和测试都是从开发人员的角度进行的。

单元测试针对一个代码片段，从软件自身的角度分析所期望的结果，而不是测试最终用户看到的结果。

14.2.1 单元测试的优点

单元测试的好处显而易见。尽早地测试代码，可以揭示出那些简单的、非结构化的开发人员进行的测试所掩盖的问题，从而提高代码质量。在功能测试期间或者在最终产品中，更高的代码质量意味着更少的 bug。

更少的 bug，可以使项目中的每一个人感到愉快，包括最终用户。

除了在项目开始就编写出更好的代码之外，单元测试的另外一个关键用途是自动回归测试。在 jQuery 项目自身中，这也是单元测试最重要的用途之一。使用自动化的单元测试可以确保新特性或者对 bug 的修正不会引入其他错误。这对于 jQuery 库的稳定性至关重要。

在 jQuery Bug 修正指南(<http://weblog.bocoup.com/javascriptjquery-bug-fixing-guide>)中，Rick Waldron 的原话为(全部字母大写)：“ALWAYS RUN THE FULL SUITE BEFORE COMMITTING AND PUSHING A PATCH!!!”——在提交和发布一个补丁之前，必须运行完整的全套测试。他已经说得很清楚了。

运行完整的测试套件并确保所有测试都能成功通过，说明对 bug 的修正添加到 jQuery 库中作为一个整体，而不是从 jQuery 库中减去该修正。

单元测试的结构可以更容易地实现重构。由于准确地知道一个特定代码单元需要通过什么样的测试，开发人员更自信地重构该单元的代码，不必时常担心漏掉某些依赖项或者其他功能。任何开发人员都不希望看到经其重写后的代码导致整个应用程序崩溃。重写后的代码如果通过了已经创建的单元测试，就可以大大减轻这种恐惧。

另外，由于良好的单元测试依赖于测试模块的独立性，因此编写单元测试的代码将强迫开发人员编写出更加松耦合(loosely coupled)的代码。开发人员要避免编写跨越多个模块的测试，就必须编写更抽象的输入和输出接口，以便在测试框架中更容易地进行模拟。

编写独立的单元测试代码还意味着：在该单元依赖的其他系统完成之前，就可以对该单元的代码进行测试和验证。开发过程中，甚至在其他模块之后才会被创建出来的情况下，只要具有接口的良好说明文档和编写良好的单元测试，开发人员就可以自信地编写该单元的代码。对于跨越多个不同组织的大型团队而言，这无疑是一个巨大的优势。一个顾问可以加入进来，编写一个他最擅长的模块，如果他的代码经过了良好的测试并且与商定的接口能正确地工作，那么理论上他就可以去干下一项工作，不必等待组织中的其余工作赶上他的进度。这并未否定集成测试的必要性，集成测试可以确保各个系统在连接起来之后，实际的系统功能与预期的一致。但是相对于缺少结构化测试方法所开发的系统，单元测试可以减少很多意外问题产生。

14.2.2 测试驱动的开发

单元测试是一种称为测试驱动的软件开发(Test-Driven Development, TDD)方法论的基础。TDD 以较小的单元测试作为开发生命周期，它鼓励极短的开发周期。开发人员将从编写一个失败的单元测试开始，随后将编写最小数量的代码以通过失败的测试。最终他将转移到下一个特性的开发，或者对代码进行重构以获得更好的质量。尽管本章并不涉及具体的 TDD 实践，但 TDD 常常与单元测试相提并论，因此指出二者的关系很重要。

总的来说，没有单元测试就无法采用 TDD 进行软件开发，但可以只使用单元测试，而不必涉足完整的 TDD 实践过程。

14.2.3 什么是一个好的单元测试

在介绍完单元测试的基本概念之后，重要的是理解如何创建一个高质量的单元测试。

一个良好的单元测试具有几个标志。下面的列表说明了一些应该遵循的原则。通常情况下，单元测试应该具有以下特性：

- **自动化**：让开发人员接受单元测试并不容易，因此单元测试应该尽可能地简单，否则没有人愿意使用它。
- **快速**：否则开发人员几乎不可能愿意使用单元测试工具。
- **集中(Focused)**：所测试的组件应该尽可能地小，以减少含糊不清之处。
- **自主性(Independent)**：可以从以下两个角度来说明，这两点都很重要：
 - **自包含(Self-contained)**：测试应该是原子化的。这意味着可能需要使用一些代码来代替外部的输入或输出。在测试期间，这些代码被称为 mock(模拟)或 stub(桩)。本章后面将更详细地介绍 mock。
 - **独立性(Standalone)**：除了没有外部依赖之外，单元测试应该可以按照任意次序运行，在次序上不必依赖于任何其他测试。

- **一致性(Consistent)**: 测试应该总是返回相同的结果。
 - **易读(Readable)**: 测试的意图和焦点应该明确, 项目后期的开发人员不需要太多熟悉时间就应该可以明白测试的意图。
 - **可维护性(Maintainable)**: 单元测试与项目是共同存在的, 因此单元测试也应该具有可维护性。使用过期的测试比根本没有测试要好, 但过期的测试也将导致自身的问题。
 - **可信赖(Trustworthy)**: 开发人员应该对单元测试的结果完全信任。
- 现在你已经了解了单元测试的要点, QUnit 这个明星该登场了。

14.3 QUnit 入门

QUnit 产生于 2008 年 5 月, 它来源于 jQuery 项目自己的 testrunner。随着 QUnit 的发展, 它从 jQuery 中剥离出来成为一个单独的项目, 形成了自己的名称、文档并最终完全从 jQuery 中独立出来。

如果你对单元测试常见的概念有所了解, 那么开始运行 QUnit 就比较简单。下面几个小节简要地介绍了 QUnit 测试的关键结构, 描述了在 QUnit 中测试代码所使用的基本断言。

14.3.1 在 QUnit 中使用 equal 测试 Hello World

下面的文件显示了一个 QUnit 的梗概。它包含了一个 jQuery 副本、一个 QUnit 脚本文件的副本、一个 QUnit CSS 文件的副本以及一些组成测试的脚本和标记代码。

在下面的脚本块中使用了 QUnit 的 `test()` 方法进行测试。顾名思义, `test()` 方法的功能是建立一个要运行的测试。它接收两个参数, 第一个参数是一个名称——本例中的 `Hello World`, 第二个参数是一个实际要测试的函数。在本例的测试函数中使用了 QUnit 的 `equal()` 函数。在单元测试的术语中, `equal` 指的是一个断言。一个断言指出期望代码执行什么样的功能。顾名思义, `equal()` 方法对“是否相等”进行测试。它接收三个参数: 第一个参数是一个实际要测试的变量, 第二个参数是断言所期望的值, 第三个参数是该断言要显示的消息。

在下面的标记代码中包含了一些 QUnit 特殊的 HTML 元素:

#qunit-header: 包含了该测试单元的名称。

#qunit-banner: 当某个测试失败时显示为红色, 如果所有测试都通过, 则显示为绿色。

#qunit-userAgent: 显示 `navigator.userAgent` 属性。

#qunit-tests: 是一个包含测试结果的容器。

#qunit-fixture: 列出要执行的单元测试。在下一小节中将详细介绍 `#qunit-fixture`。



```
<!DOCTYPE html>
<html>
<head>
```

```

<script src="http://code.jquery.com/jquery-1.7.1.js"></script>
<link rel="stylesheet"
  href="http://code.jquery.com/qunit/git/qunit.css"
  type="text/css"
  media="screen" />
<script src="http://code.jquery.com/qunit/git/qunit.js"></script>
<script>
$( function(){
  test("Hello World", function() {
    var message = "hello world";
    equal( message,
      "hello world",
      "We expect the message to match"
    );
  });
});</script>
</head>
<body>
  <h1 id="qunit-header">QUnit example</h1>
  <h2 id="qunit-banner"></h2>
  <div id="qunit-testrunner-toolbar"></div>
  <h2 id="qunit-userAgent"></h2>
  <ol id="qunit-tests"></ol>
  <div id="qunit-fixture">Test Markup</div>
</body>
</html>

```

代码片段 *simple-test.html*

运行上面的测试，输出结果如图 14-1 所示。



图 14-1

如果看到上图的输出结果，则说明 QUnit 已经运行成功。

1. xUnit 与 QUnit 的差别

尽管二者的名称都隐含了单元测试的意思，然而请注意 QUnit 并不是 xUnit 家族的一员。

xUnit 这个名称表示的是一个测试框架家族，它们的根源可以追溯到 Kent Beck 在一篇名为 Simple Smalltalk Testing: With Patterns(<http://www.xprogramming.com/testfram.htm>)的论文中描述的设计。最初在 Smalltalk 中的实现是 SUnit，该设计现在已经移植到多种其他语

言之中, 包括 Java(JUnit)、C++(CppUnit)和.NET(NUnit)。

虽然 QUnit 与 xUnit 分享了一些基本的概念, 比如断言。但 xUnit 的设计与 QUnit 之间存在着显著差别。例如, 在前面的例子中见到的 `equal()` 方法的两个参数: `actual` 和 `expected`, 在 xUnit 的设计中, 这两个参数的顺序刚好相反。

另外 xUnit 和 QUnit 还有着完全不同的断言名称。比如对于前面例子中 QUnit 的 `equal` 断言, 在 xUnit 中与之类似的断言是 `assertEquals`。二者的功能相同, 在语法上有着细微的区别。另外一个区别是 xUnit 中具有某些断言在 QUnit 中并没有。

如果你对 xUnit 家族的单元测试工具并不熟悉, 那么这当然不会成为什么问题。如果你使用过 xUnit 工具, 那么应该注意一下二者的区别, 以免在使用 QUnit 时出现错误。

了解了这一潜在的问题之后, 让我们继续学习 QUnit。

14.3.2 一个失败的 QUnit 测试

前面已经介绍过一个成功通过的测试, 下面介绍一个失败的测试。使用与前面例子相同的标记代码, 使用 `equal` 断言运行下面的测试:



可从
Wrox.com
下载源代码

```
test("Hello World", function() {
    var message = "Goodbye cruel world";
    equal( message, "hello world", "We expect the message to match " );
});
```

代码片段 *failing-test.html*

从图 14-2 中可以看到, 与成功通过的测试相比, 失败的测试提供了更多的有用信息。首先, 在 `equal` 方法第 3 个参数中定义的消息将显示出来, 并列出了所测试变量的期望值和实际值。在本例中, 变量 `message` 的期望值为 `hello world`, 实际值为 `Goodbye cruel world`。另外, 它指出了期望值与实际值的差异, 并指出是哪一行代码导致了该错误。

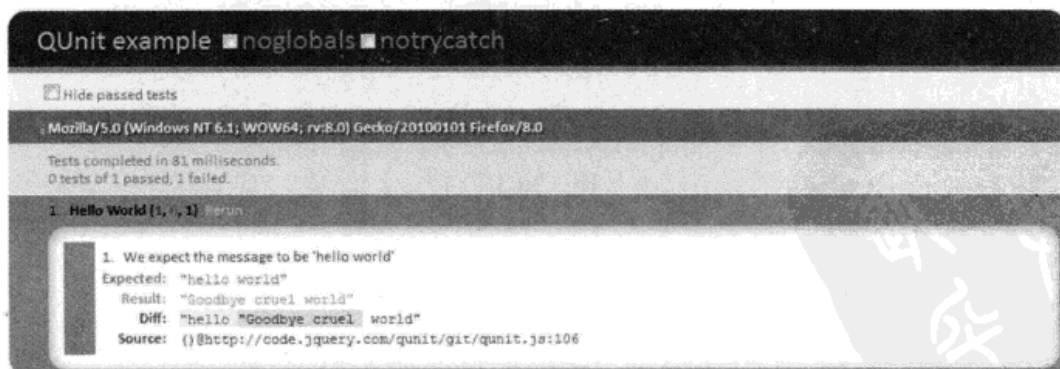


图 14-2

14.3.3 使用 `ok` 测试真伪

在某些情况下, `ok()` 方法可以作为一个更好的 Hello World 版示例, 因为它是最基本的断言。`ok()` 方法接收两个参数: 第一个参数是一个要进行测试的布尔陈述; 第二个参数是

当测试失败时要显示的消息。如果所测试的布尔陈述为真(true)则测试通过。下面的代码示例演示了如何使用 `ok()` 方法测试布尔值。它使用 jQuery 的 `$.isArray()` 方法测试 `val()` 函数的返回值是否为真。



可从
Wrox.com
下载源代码

```
test("Truthiness is golden", function() {
    var val = function(){
        var arr = [1,2,3,4,5]
        //执行某些操作
        return arr;
    }
    ok( $.isArray(val()), "We're expecting an array as a return value.");
});
```

代码片段 ok.html

14.3.4 设置预期的断言数量

为测试设置预期的断言数量是一个良好的习惯。这可以告知 QUnit “期望执行 x 个断言”。如果 QUnit 没有精确地看到 x 个断言，那么整个测试将失败，即便所有这些断言都通过了测试。

从最简单的输入错误，到与应用程序流有关的更复杂的问题，出于很多原因，测试可能不会运行。在学习后面的小节中关于异步测试的内容时，可以看到一个基本的例子，它说明了为什么设置一个预期的断言数很重要。

在 QUnit 中可以用两种方式来设置预期的断言数。第一种方法是在 `test()` 方法体内调用 `expect()` 方法。下面的例子说明了如何使用 `expect()` 方法。我们期望执行两个断言，但只有一个断言存在，因此整个测试失败。



可从
Wrox.com
下载源代码

```
test("Hello World", function() {
    expect(2)
    var message = "hello world";
    equal( message, "hello world", " We expect the message to match." );
});
```

代码片段 expect.html

图 14-3 显示了该测试的结果。即使所有断言都通过了测试，但是由于没有达到期望的断言数量，因此整个测试失败了。

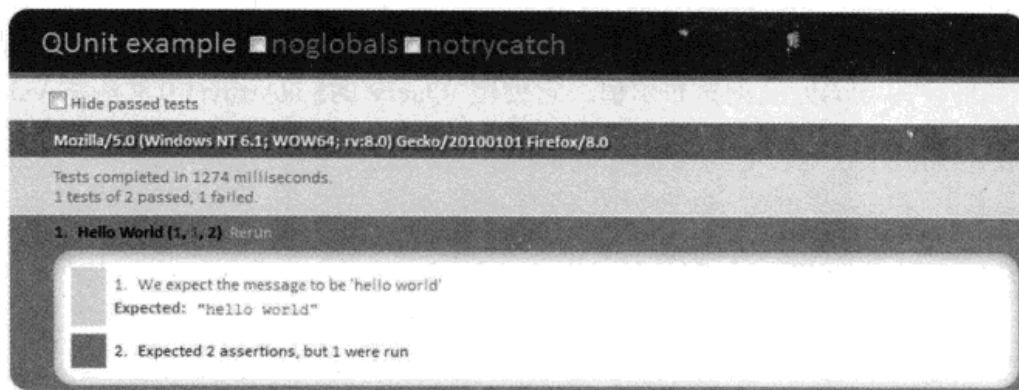


图 14-3

除了使用 `expect()` 方法之外，还可以用 `test()` 方法可选的第 2 个参数来设置预期的断言数，比如下面的代码：



可从
Wrox.com
下载源代码

```
test("Hello World", 2, function() {
    var message = "hello world";
    equal( message, "hello world", "We expect the message to match" );
});
```

代码片段 `expect.html`

尽管这两种设置预期断言数的方法都是有效的，在较大的测试过程中都可以看到这两种方法的应用，在 `test()` 方法体内显式调用 `expect()` 方法更加简洁，也更容易阅读。笔者更喜欢采用 `expect()` 方法，而不是在 `test()` 方法中设置第 2 个参数。

14.3.5 其他断言

表 14-1 列出了 QUnit 中所有有效的断言。其中绝大多数断言的语法都很简单，与这一小节中介绍的其他断言的语法非常类似。这些断言要么是否定性断言，比如 `notEqual` (`equal` 断言的否定)；要么提供了更高准确度(或不同层次)上的断言，比如 `deepEqual` 和 `strictEqual` 断言。请熟悉列表中的这些断言，这些断言有助于你编写出精确度更高的测试。

表 14-1 QUnit 断言

断 言	参 数	描 述
Ok	state、message	布尔断言。如果 state 为真则通过测试
equal	actual、expected、message	比较断言。将 actual 与 expected 进行比较。如果 actual 等于(=)expected 则通过测试
notEqual	actual、expected、message	比较断言。将 actual 与 expected 进行比较。如果 actual 不等于 expected 则通过测试
deepEqual	actual、expected、message	深度递归比较断言。递归比较对象。如果有某一个属性不相等则测试失败

(续表)

断 言	参 数	描 述
notDeepEqual	actual、expected、message	深度递归比较断言。递归比较对象。与 deepEqual 正好相反。如果有某一个属性不相等则通过测试
strictEqual	actual、expected、message	比较断言。将 actual 与 expected 进行比较。如果 actual 与 expected 严格相等(===)则通过测试
notStrictEqual	actual、expected、message	比较断言。将 actual 与 expected 进行比较。如果 actual 与 expected 严格不相等则通过测试
Raises	block、expected、message	该断言测试代码是否抛出一个错误

14.3.6 测试 DOM 元素

如果正在执行任何的 DOM 操作(由于这是一本关于 jQuery 的书,因此你很可能需要执行 DOM 操作),那么可能需要测试输出。QUnit 提供了一个界面用于测试 DOM 操作。要测试 DOM 元素,只需要简单地将标记代码放在#qunit-fixture 中,然后执行 DOM 操作,并使用断言来测试 DOM 的状态。QUnit 将在每次测试后重置#qunit-fixture。

下面的代码演示了一个简单的例子:



可从
Wrox.com
下载源代码

```
<!DOCTYPE html>
<html>
<head>
<script src="http://code.jquery.com/jquery-1.7.1.js"></script>
<link rel="stylesheet"
      href="http://code.jquery.com/qunit/git/qunit.css" />
<script type="text/javascript"
src="http://code.jquery.com/qunit/git/qunit.js"></script>
<script>
$(function() {
    test("DOM", function() {
        expect(3);
        $test = $("#dom-test");
        $test.css("width", "200px").text("Testing the DOM").show();
        equal( $test.css("width"), "200px", "200px!" );
        equal( $test.text(), "Testing the DOM", "We're testing the DOM" );
        equal( $test.css("display"), "block", "display:block" );
    });
});
</script>
</head>
<body>
```



```

<h1 id="qunit-header">QUnit example</h1>
<h2 id="qunit-banner"></h2>
<div id="qunit-testrunner-toolbar"></div>
<h2 id="qunit-userAgent"></h2>
<ol id="qunit-tests">
</ol>
<div id="qunit-fixture">
  <div id="dom-test" style="display:none"></div>
</div>
</body>
</html>

```

代码片段 dom-test.html

jQuery 项目包含了很多示例。由于 jQuery 可以执行大量的 DOM 操作，在项目中需要确保 DOM 操作可靠地执行，因此需要使用大量的断言来测试 DOM 的状态。

14.3.7 使用 noglobals 和 notrycatch

在前面的例子中，还有两个值得注意的选项。在每一个测试中，这两个选择显示为两个复选框 noglobals 和 notrycatch，如图 14-4 所示。

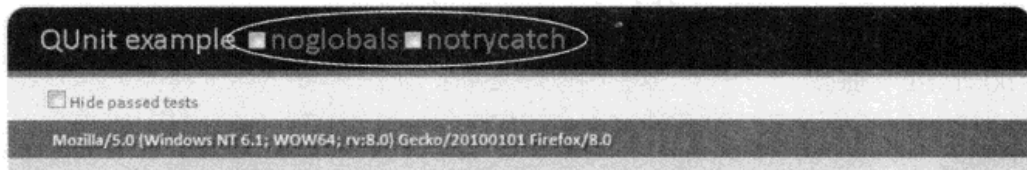


图 14-4

这两个选项都用于设置页面级的标志，它们将改变该页面上执行的所有测试的行为。

如果在运行的代码中引入了一个新的全局变量，那么 noglobals 将失败。如果你正在用 ES5 strict 模式编写代码，或者正在用 JSHint(<http://www.jshint.com/>)检查代码的质量，那么这应该不成问题，但是对于全局变量可能会潜入测试阶段的情况下，这一检查还是非常有用的。

下面的例子很简单，它显示了一个将触发 noglobals 标志的测试，相应的错误消息将显示在测试工具中，如图 14-5 所示。

```

test("Hello World", function() {
  message = "hello world";
  equal(message, "hello world", "We expect the message to be 'hello world'");
});

```

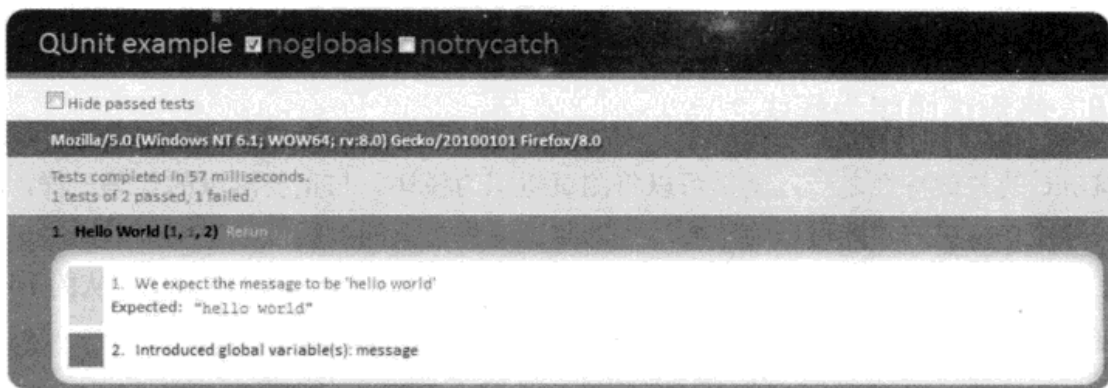


图 14-5

设置 `notrycatch` 标志将在没有 `try-catch` 块包围的情况下运行 QUnit。这实际上停止了 QUnit 的测试，以便更深入地对问题进行调试。下面的例子说明了 `notrycatch` 标记的应用：

```
test("No Try-Catch", function() {
    //下面的$$将导致一个异常
    var test = $$ (this).isGonnaExplode();

    equal( test , "Return Value of Doom", "We're looking for a cool return
        value" );
});
```

图 14-6 显示在 Firebug 中显示的异常。如果没有设置 `notrycatch`，那么 Firebug 的控制台将是空的，错误将被记录在测试工具中。

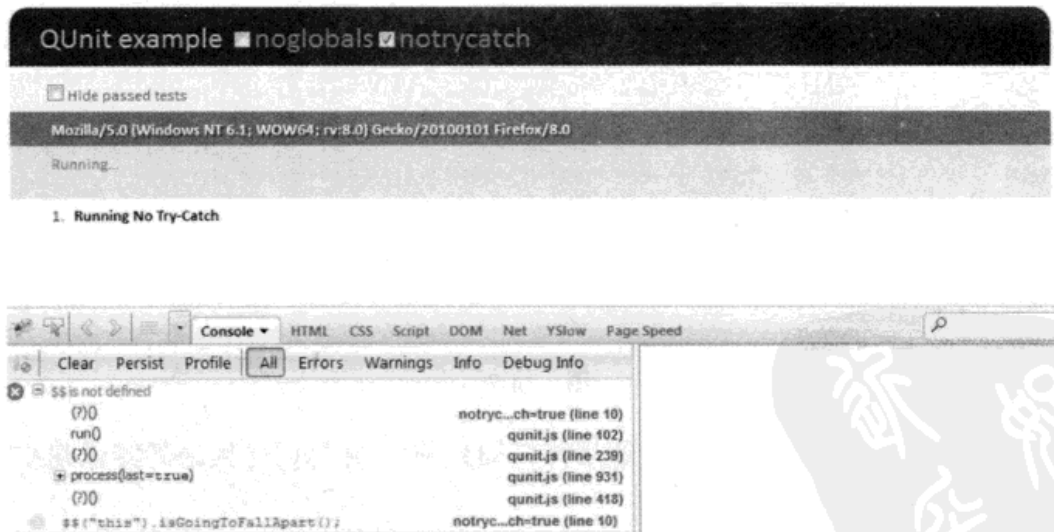


图 14-6

除了通过在 QUnit 中选中相应的复选框来运行测试之外，还可以在 URL 后添加一个查询字符串参数来运行测试。例如，要在 `noglobals` 状态下运行测试，可以使用下面的 URL：

```
http://example.com/tests/tests.html?noglobals=true
```

如果在一个大型应用程序中要进行多个测试，这是一个方便的特性。

14.3.8 将测试组织为模块

QUnit 的关键特性之一，就是可以将测试组织为模块。与正要测试的代码一样，将代码组织为独立模块的功能可以为正要运行的测试提供更大的灵活性。

更加特定的方法或特性对多个测试进行分组，这是模块的一个基本用法。从图 14-7 中对 jQuery 核心库的测试组件可以看到，该项目被划分为 17 个独立的模块。

Latest commit to the master branch

Reformat jshint errors to be readable; make post-compile js write dir... *

rwidm authored December 15, 2011
timmywil committed December 15, 2011

commit f724bc6c92

jquery / test / unit

name	age	message	history
ajax.js	December 06, 2011	Fix #10466. jQuery.param() should treat object-wrapped primitives as ... [rwidm]	
attributes.js	December 06, 2011	Fix #5571. Setters should treat 'undefined' as a no-op and be chainable. [gibson042]	
callbacks.js	November 06, 2011	Fix #10691. Remove all instances of equals() and same(), as these are ... [mikesherov]	
core.js	December 06, 2011	Refine the jQuery.isWindow check. [rafBM]	
css.js	December 12, 2011	When the width/height computed unit is not pixels, return that instea... [timmywil]	
data.js	December 06, 2011	Fix #5571. Setters should treat 'undefined' as a no-op and be chainable. [gibson042]	
deferred.js	November 08, 2011	Have Deferred always return the object onto which it is currently att... [jaubourg]	
dimensions.js	December 06, 2011	Fix #5571. Setters should treat 'undefined' as a no-op and be chainable. [gibson042]	
effects.js	December 08, 2011	Fix #8498. Add cssHooks[prop].expand for use by animate(). [mikesherov]	
event.js	December 13, 2011	Fix #11021. There should be no mangling of the "hover" namespace. [dmethvin]	
exports.js	November 14, 2011	Landing pull request 586. Create exports.js for exporting jQuery to w... [jrburke]	
manipulation.js	December 06, 2011	Fix #5571. Setters should treat 'undefined' as a no-op and be chainable. [gibson042]	
offset.js	December 06, 2011	Fix #5571. Setters should treat 'undefined' as a no-op and be chainable. [gibson042]	
queue.js	December 06, 2011	Fix #5571. Setters should treat 'undefined' as a no-op and be chainable. [gibson042]	
selector.js	October 13, 2011	Update sizzle; Add sizzle cache collision iframe test. Fixes #8539. [timmywil]	
support.js	November 18, 2011	No global vars allowed. Declare 'body' in support. [timmywil]	
traversing.js	December 12, 2011	Use Sizzle Expr.match.globalPOS for identifying POS selectors in trav... [timmywil]	

图 14-7

另外，还可以在运行时过滤模块(随后将详细介绍)，并为运行模块的测试建立特定的环境变量。

下面的代码显示了一个简化的模块，它定义了两个新特性以建立用户运行环境：setup 和 teardown。这两个方法将分别在模块测试之前和之后运行，就为类似方法建立测试环境而言，这种方式提供了很大的灵活性。

在下面的例子中，为了测试整个模块，建立了一个名为 arr(一个数组)的变量。另外，还将 arr.length 的值保存在一个名为 control 的变量中，在测试过程中如果数组发生了改变，可以用该变量进行对比。在模块自身中，包含了两个测试。第一个测试包含两个 ok 断言，

第二个测试则包含了一个 `deepEqual` 断言。它们都是对 `Array` 的不同方面进行测试。



```
module("fun with arrays", {
  setup: function() {
    this.arr = [1,2,3,4,5];
    this.control = this.arr.length;
  },
  teardown: function() {
    equal(this.arr.length, this.control, "Just Checking to see if
    it's still an Array");
  }
});

test("Truthiness is golden", function() {
  expect(3);
  var bool = $.isArray(this.arr);

  ok( bool, "We're expecting an array" );

  bool = this.arr.indexOf(5);
  ok(bool, "We expect 5 to be in the Array");
});

test("Getting Deep", function() {
  expect(2);
  deepEqual(this.arr, [1,2,3,4,5]);
});
```

代码片段 `module.html`

运行这些测试，输出结果如图 14-8 所示。可以看到，这些测试都嵌套在 `fun with arrays` 模块名称之下。

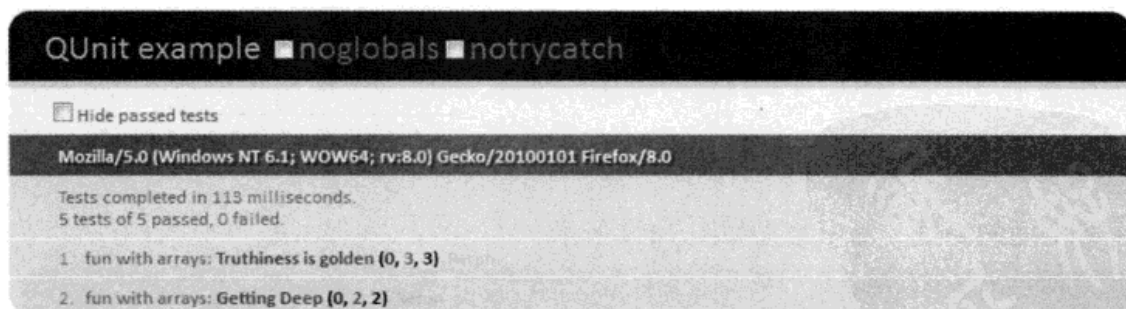


图 14-8

这种模块测试方法，允许开发人员对特定的网站部分或特性进行测试，而不必运行一个完整的全套测试。如果你正在开发一个较大的、具有很多关联测试的项目，这将是一个非常有用的特性。

1. 使用 URL 参数过滤测试

可以在查询参数中设置 `noglobals` 和 `notrycatch` 标志, 采用类似的办法, 也可以在查询参数中设置参数以过滤测试。只需要简单地在 URL 之后添加 `?filter=filename`, QUnit 将只运行名称中包含 `filename` 的测试。采用这种方式, 可以根据模块名称或测试名称进行测试, 并允许在执行测试时轻松地瞄准特定的特性。

在修正 jQuery 核心库的 bug 时, 使用 URL 参数过滤测试的好处是显而易见的。尽管在提交一个关于 jQuery bug 的问题之前要求进行完整的全套测试, 但我们并不想测试整个 jQuery 库, 只需要对要修正的特定特性进行测试即可。

在开发过程中, 通过使用 URL 参数过滤测试, 可以仅针对正在开发的关键部分进行测试, 之后在发布(这也许需要点英雄气概)对 bug 的修正之前, 只需要运行一次全面的测试即可。

14.4 异步测试

有时需要控制测试的执行流。一个显而易见的例子是对 Ajax 请求的测试。比如在下面的代码示例中, 可以看到在测试中包含了一个 Ajax 请求。该 Ajax 请求的回调函数中包含了两个断言: 一个 `ok` 断言和一个 `equal` 断言。



可从
Wrox.com
下载源代码

```
test("stops & starts", function() {
    expect(2);

    var url = "/data/json/";

    $.getJSON(url, {name: "Testing"}, function(data) {
        ok(data, "data is returned from the server");
        equal(data.name, "Testing", "We're Testing");
    });
});
```

代码片段 `broken-ajax-test.html`

运行上面的测试, 输出结果如图 14-9 所示。



图 14-9

显然这并不是我们所需要的结果。

由于 Ajax 请求是异步的, 因此这两个测试都不会运行。采用这种方式进行测试无法捕获任何错误。幸运的是, QUnit 提供了两个控制测试执行流的方法: `start()`方法和 `stop()`方法。重新编写上面的测试, 可以使其按照我们所期望的方式工作。在发起 Ajax 请求之前, 测试将被 `stop()`方法暂停, 在回调函数的末尾, `start()`方法将重新启动该测试。

```
test("asynch", function() {
    expect(2);

    stop();

    $.getJSON("/data/json/", function(data) {
        ok( data, "We expect to get data from the server");
        equal(data.name, "Testing", "We expect the name property to match");
        start();
    });
});
```

代码片段 `start-stop.html`

运行该测试, 可以产生我们所期望的结果。从图 14-10 中可以看到, 两个测试都已经运行并通过了测试。

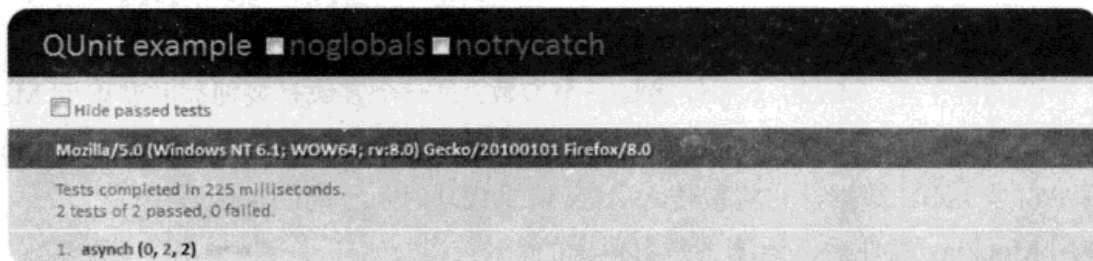


图 14-10

14.4.1 使用 `asyncTest`

与便利的 `$.Ajax()`方法类似, QUnit 也具有一个便利方法用于常见的异步测试模式, 它可以启动一个测试、然后在一个异步行为中停止该测试。

在下面的代码示例中, 使用便利的 `asyncTest()`方法重写了前面的例子:



可从
Wrox.com
下载源代码

```
asyncTest("asynch", function() {
    expect(2);

    $.getJSON("/data/json/", function(data) {
        ok( data, "We expect to get data from the server");
        equal(data.name, "Testing", "We expect the name to match");
    });
});
```



```

        start();
    });
    });

```

代码片段 `asynctest.html`

强烈建议采用 `asyncTest()` 方法执行异步测试。它可以保证异步测试正确地建立，并使普通测试和异步测试的代码很容易区分。采用 `asyncTest()` 方法可以使代码具有更好的可读性。其他开发人员不必从 `test()` 方法体内阅读任何断言或其他逻辑，就可以清楚地知道这是一个异步测试。

14.4.2 模拟 Ajax 请求

虽然可以使用实际的数据来测试 Ajax 请求，但并不建议这样做。本章前面曾经介绍过，单元测试的基础之一就是测试应该是自包含的。为了实现这一要求，最佳实践是模拟 Ajax 请求。

QUnit 中已经提供了一个实现该功能的插件：`Mockjax`。`Mockjax` 来自于著名的 `appendto` (<https://github.com/appendto/jquery-mockjax>)，该插件可以捕获 URL。这意味着在请求发送到传统的 Ajax 方法之前，`Mockjax` 可以拦截 Ajax 请求并处理对该请求的响应。`Mockjax` 可以建立一个可测试的响应，并允许根据自己的设计自定义该响应的 HTTP 头。

可以用 `Mockjax` 重写前面的例子，以便更好地遵循单元测试的最佳实践。下面的例子演示了如何使用 `Mockjax` 插件，建立一个正确的、可测试的 Ajax 请求。

在下面的例子中，`Mockjax` 插件接收三个参数：第一个参数是要捕获并映射到 `Mockjax` 的 url，第二个参数是用于模拟延迟的响应时间 `responseTime`，第三个参数是 `responseText`，在本例中它是 JSON 数据，它将被传入 Ajax 请求成功时执行的回调函数中。

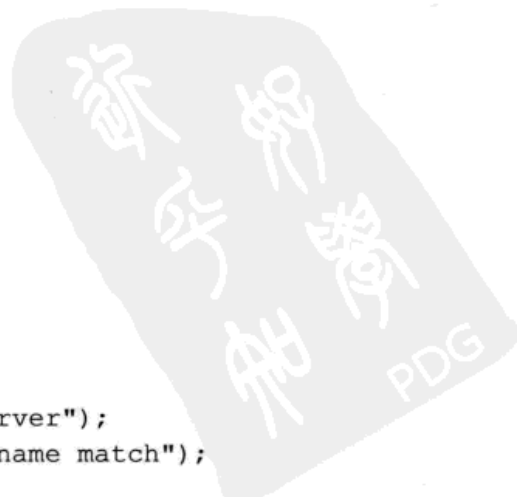
使用 `Mockjax` 插件后，该测试是自包含的。



```

test("asynch", function() {
    expect(2);
    $.mockjax({
        url: '/data/json/',
        responseTime: 100,
        responseText: {
            status: 'success',
            name: 'Testing'
        }
    });
    stop();
    $.getJSON("/data/json/", function(data) {
        ok( data, "We expect to get data from the server");
        equal( data.name, "Testing", "We expect the name match");
        start();
    });
});

```



```
});
});
```

代码片段 `mockjax.html`

即使没有 Internet 连接或者一个完善的数据服务, 该测试也能够通过。这种方法允许开发人员对 Ajax 请求的代码进行测试, 不必担心系统其他部分的状态如何。如果你正在一个大型开发团队中工作, 那么 Mockjax 插件的这种模拟功能会令你如获至宝, 它允许你编写并测试一个接口, 而不必担心团队中的其他人是否已经完成了该接口。

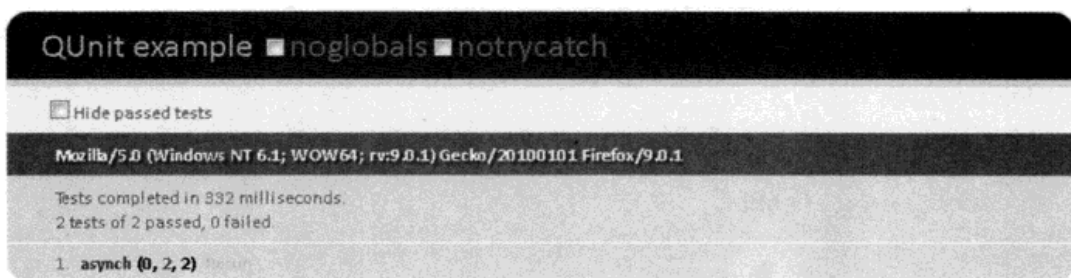


图 14-11

1. 其他 Mockjax 特性

由于现代 Web 应用程序开发需要处理的问题远比一个简单的 JSON 请求复杂得多, Mockjax 具备足够的特性以处理绝大多数 Ajax 请求的情况。Mockjax 的更多特性值得学习, 因为在测试中我们不知道会面对什么样的情况。下面先介绍两个最值得学习的 Mockjax 特性。

2. 使用 Mockjax 数据代理

Mockjax 最优秀的特性之一, 就是可以创建一个数据代理。当建立一个简单的测试时, 在测试中列出一组数据元素是得很方便的, 但如果试图测试一个完整的界面, 在 Mockjax 插件的调用中包含一个较大的数据结构就显得不太灵活。为了解决这一问题, Mockjax 提供了一个 proxy 选项, 用于设置一个数据代理, 它指向一个作为 Ajax 请求数据源的静态文件。所需做的工作, 只需要在建立 Mockjax 插件时添加一个 proxy 参数即可, 不必包含数据源文本本身。下面是一个最简单的例子:

```
$.mockjax({
  url: '/data/json',
  proxy: '/mock/data.json'
});
```

采用这样的办法, 开发人员就可以使用复杂的数据结构, 而不必担心在测试框架中潜入这些复杂的结构。请注意, 如果在低版本浏览器中测试, 当确实需要访问 JSON.stringify 时, 可能需要使用到 json2.js(<https://github.com/douglascrockford/JSON-js>)。

3. 添加额外的 HTTP 头

Mockjax 还允许开发人员设置额外的 HTTP 头。对于高级 Ajax 程序设计这一点是至关重要的。可以在建立 Mockjax 对象时定义一些标准的设置，比如 HTTP 的 status 和 contentType。其他参数则可以使用一个可选的 headers 属性进行设置，headers 属性可以包含一系列的 name/value 对，用于表示我们需要的 HTTP 头。下面的例子设置了三个 HTTP 头属性：

```
$.mockjax({
  url: '/data/json',
  //没有找到文件!
  status: 40,
  contentType: 'text/json',
  headers: {
    secretSauce: 'shhh'
  }
});
```

14.5 综合示例

这里再强调一下，jQuery 测试套件提供了一些如何组织 QUnit 测试的教程。实际上，没有什么比查看 QUnit 具体的实现更好的教程了，它是最全面、最成熟的 QUnit 测试案例的集合。

前面曾经介绍过，jQuery 测试套件被划分为 17 个模块，这些模块代表了 jQuery 库中更小的功能划分粒度，比如 data、css 和 deferred。在这些模块中包含了多种测试，每一个测试都包含了多个断言。要对在实际的项目中如何将这此模块组织起来有所了解，请查看下面的代码示例中列出的经过大量精简的核心测试模块。其中删除了很多测试，并对几乎所有断言进行了注释。但从该示例中可以直观地看出测试的结构、粒度和涉及的测试方法。

从下面的例子中可以看到，特殊的特性被单独测试，并且每个测试具有多个断言。另外请注意，在 jQuery.merge() 和 jQuery('html', context) 方法中报告了 bug 的数量。在报告了问题之后，创建了测试用例，并修正了代码。这些测试用例保留下来，可以为数月甚至数年之后才修改的 bug 提供保证。另外，简单地提醒一下，QUnit 就是 JavaScript，在 jQuery.camelCase() 测试中可以看到动态创建的测试，它使用一个 Array 作为测试对象，然后用 jQuery.each 遍历数组，对 7 个期望断言中的每一个断言进行测试。



可从
Wrox.com
下载源代码

```
module("core", { teardown: moduleTeardown });
test("Basic requirements", function() {
  expect(7);
  ok( Array.prototype.push, "Array.push()" );
```



```

    ok( Function.prototype.apply, "Function.apply()" );
    ok( document.getElementById, "getElementById" );
    ok( document.getElementsByTagName, "getElementsByTagName" );
    ok( RegExp, "RegExp" );
    ok( jQuery, "jQuery" );

    ok( $, "$" );
  });

  test("jQuery()", function() {
    expect(29);
    //29 assertions
  });

  test("selector state", function() {
    expect(31);
    //31 assertions
  });

  test("jQuery('html')", function() {
    expect(18);
    //18 assertions
  });

  test("jQuery('html', context)", function() {
    expect(1);

    var $div = jQuery("<div/>")[0];
    var $span = jQuery("<span/>", $div);
    equal($span.length, 1, "Verify a span created with a div context works, #1763");
  });

  test("first()/last()", function() {
    expect(4);

    var $links = jQuery("#ap a"), $none = jQuery("asdf");

    deepEqual( $links.first().get(), q("google"), "first()" );
    deepEqual( $links.last().get(), q("mark"), "last()" );

    deepEqual( $none.first().get(), [], "first() none" );
    deepEqual( $none.last().get(), [], "last() none" );
  });

  test("map()", function() {
    expect(8);
    //8 assertions
  });

  test("jQuery.merge()", function() {

```

```
expect(8);

var parse = jQuery.merge;

deepEqual( parse([],[]), [], "Empty arrays" );

deepEqual( parse([1],[2]), [1,2], "Basic" );
deepEqual( parse([1,2],[3,4]), [1,2,3,4], "Basic" );

deepEqual( parse([1,2],[]), [1,2], "Second empty" );
deepEqual( parse([], [1,2]), [1,2], "First empty" );

// Fixed at [5998], #3641
deepEqual( parse([-2,-1], [0,1,2]), [-2,-1,0,1,2],
"Second array including a zero (falsy)");

// After fixing #5527
deepEqual( parse([], [null, undefined]), [null, undefined],
"Second array including null and undefined values");
deepEqual( parse({length:0}, [1,2]), {length:2, 0:1, 1:2},
"First array like");
});

test("jQuery.extend(Object, Object)", function() {
    expect(28);
    //28 assertions
});

test("jQuery.each(Object,Function)", function() {
    expect(14);
    //14 assertions
});

test("jQuery.sub() - Static Methods", function(){
    expect(18);
    //18 assertions
});

test("jQuery.sub() - .fn Methods", function(){
    expect(378);
    //378 assertions
});

test("jQuery.camelCase()", function() {

    var tests = {
        "foo-bar": "fooBar",
        "foo-bar-baz": "fooBarBaz",
        "girl-u-want": "girlUWant",
    }
```

```

    "the-4th-dimension": "the4thDimension",
    "-o-tannenbaum": "OTannenbaum",
    "-moz-illa": "MozIlla",
    "-ms-take": "msTake"
  };

  expect(7);

  jQuery.each( tests, function( key, val ) {
    equal( jQuery.camelCase( key ), val, "Converts: " + key + " => " + val );
  });
});

```

代码片段 jquery-suite.html

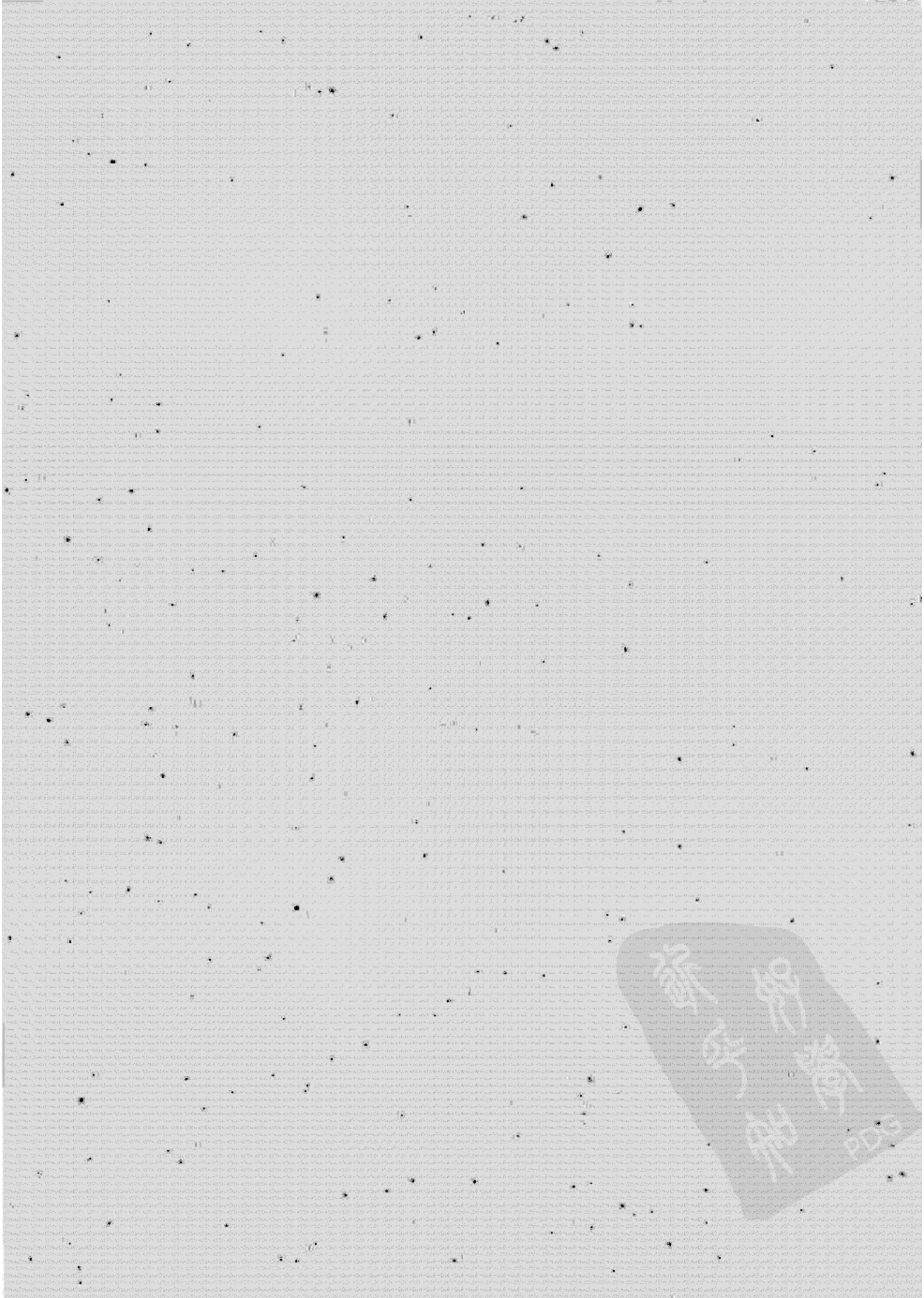
14.6 小结

本章介绍了单元测试的基本概念,还介绍了使用 QUnit 编写单元测试所需的基础知识。本章不但介绍了基本的断言,还介绍了用于测试 DOM 操作和 Ajax 交互的方法。你可以根据需要采用这些方法来测试自己的应用程序。

这一章也是本书的结尾。经过本书的学习,你已经深入地理解了 jQuery 具备的功能。本书的前半部分主要介绍了一些 jQuery 的核心概念,后半部分的章节则主要介绍了 jQuery 的一些高级主题,经过本书的学习,相信你已经成为一个既自信又能干的 jQuery 程序员。在超越了简单的 DOM 操作、预设的动画和实现插件这些知识之后,你应该已经准备好与 JavaScript 世界中的忍者和高手并肩作战了。也许你正在准备深入地开发一些自己的 jQuery 插件、增强并扩展 jQuery 核心功能以满足新的用途,或者你正准备使用 QUnit,通过形式化的测试框架提高 JavaScript 的水平和严谨程度。无论 you 从本书中学习了什么内容,最重要的一点是任何库或工具的价值都是由使用者生成的。深入学习本书的内容将有助于你提高 jQuery 的使用价值。

亲爱的你,充分发挥 jQuery 的威力吧!





附录

本书中使用的插件

Firebug Firebug 是一个 Firefox 插件，它提供了多种 Web 开发工具，用于编辑、调试和监视 CSS、HTML 和 JavaScript。

<http://getfirebug.com/>

FireQuery FireQuery 是一个 Firebug 扩展，用于 jQuery 开发。

<http://firequery.binaryage.com/>

Modernizr Modernizr 是一个小型 JavaScript 库，它提供了特性检测、工具和低版本 Internet Explorer 浏览器中的元素通用化，从而帮助开发人员充分利用 HTML5 和 CSS3 的新功能。

<http://www.modernizr.com/>

QUnit QUnit 是一个强大的、易于使用的 JavaScript 测试套件，jQuery 项目使用 QUnit 来测试它的代码和插件。

<http://docs.jquery.com/QUnit>

Data Link Plugin Data Link plugin 可以将对象链接起来，当其中一个对象的一个或多个属性发生改变时，更新另外一个对象的相应属性。

<https://github.com/jquery/jquery-datalink>

Globalization Plugin Globalization 插件支持复杂的与文化区域有关的数字和日期的解析和格式化。包括数百种不同语言和国家的传统文化信息，它还包括一个可扩展的本地化系统。

<https://github.com/jquery/globalize>

HTML5-Form Plugin 该插件为所有版本的 Internet Explorer 和 Firefox 浏览器添加了 HTML5 特有的表单验证功能。

<http://www.matiasmancini.com.ar/jquery-plugin-ajax-form-validation-html5.html>

Easing Plugin Easing 插件是一个 jQuery 插件，它为动画提供了高级 easing 选项。

<http://gsgd.co.uk/sandbox/jquery/easing/>

Metadata Plugin Metadata 插件可以从类、子元素和属性包括 HTML5 的 data 属性中提取元数据。

<http://archive.plugins.jquery.com/project/metadata>

JsViews JsViews 插件提供了数据驱动的视图，它构建于 JsRender 模板之上。

<https://github.com/BorisMoore/jsviews>

JsRender JsRender 提供了高性能、纯粹基于字符串的 JavaScript 模板，它的呈现不需要 DOM，也不依赖于 jQuery。

<https://github.com/BorisMoore/jsrender>

jQuery Templates Plugin 该插件集成到了 jQuery 中，它要为呈现在 HTML DOM 中的数据或数组提供模板。

<http://api.jquery.com/category/plugins/templates/>

Mockjax Plugin 该插件为标准 Ajax 行为流中的请求/响应提供了 Ajax 功能的模拟。

<https://github.com/appendto/jquery-mockjax>

JsFiddle 该插件为 HTML、CSS 和 JavaScript 程序提供了一个基于 Web 的测试环境。这是测试本书中代码示例的一个好办法。

<http://jsfiddle.net>



北航

C1641409